NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ROLE-BASED ACCESS CONTROL FOR LOOSELY COUPLED DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

by

Greg Nygard
Faouzi Hammoudi

March 2002

Thesis Advisor:	James Bret Michael
Thesis Co-Advisor:	John Osmundson

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
29 Mar 2002	N/A	-

Title and Subtitle Role-Based Access Control for Loosely Coupled Distributed Database Management Systems	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Hammoudi, Faouzi Nygard, Greg	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Naval Postgradaute School Monterey, California	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes The original document contains color images.	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 132	

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters)
ROLE-BASED ACCESS CONTROL FOR LOOSELY COUPLED DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

5. FUNDING NUMBERS

6. AUTHOR(S) Greg Nygard
Faouzi Hammoudi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Much of the work to date to apply Role-Based Access Control (RBAC) to database management systems has focused on single database systems or an integrated distributed database system. For situations where the need exists to consolidate multiple independent databases, and where the direct integration of the databases is neither practical nor desirable, the application of RBAC requires that policy be enforced via a method that is distinct from the databases. The method must provide for the verification of the RBAC policy, while allowing for the independence of the various databases on which the policy is enforced. This paper proposes a model for an application that provides for a web-based interface for users to be granted access to data held in various independent databases. The application enforces a strict RBAC policy on a well-defined set of accesses, while alleviating the need for users to have a separate account on each of the databases.

14. SUBJECT TERMS Database Management, Distributed Computing, Role-based Access Control, Security Policy

15. NUMBER OF PAGES 132

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ROLE-BASED ACCESS CONTROL FOR LOOSELY COUPLED DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

Greg L. Nygard
Lieutenant, United States Navy
B.S., University of Arizona, 1993

Faouzi Hammoudi
Captain, Tunisian Air Force
Laureate in Electronic Engineering, University of Naples "Federico II", Italy, 1992

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT
AND
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2002

Authors: Greg L. Nygard

Faouzi Hammoudi

Approved by: James Bret Michael, Thesis Advisor

John Osmundson, Thesis Co-Advisor

Dan Boger, Chairman, Information Technology Department

CDR Chris Eagle, Chairman, Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Much of the work to date to apply Role-Based Access Control (RBAC) to database management systems has focused on single database systems or an integrated distributed database system. For situations where the need exists to consolidate multiple independent databases, and where the direct integration of the databases is neither practical nor desirable, the application of RBAC requires that policy be enforced via a method that is distinct from the databases. The method must provide for the verification of the RBAC policy, while allowing for the independence of the various databases on which the policy is enforced. This paper proposes a model for an application that provides for a web-based interface for users to be granted access to data held in various independent databases. The application enforces a strict RBAC policy on a well-defined set of accesses, while alleviating the need for users to have a separate account on each of the databases.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS

DBMS – Database Management System

DRBAC – Database Role-based Access Control

DSD – Dynamic Separation of Duty

HSQLDB – Hypersonic SQL Database

JCA – Java Cryptography Architecture

JCE – Java Cryptography Extension

JDBC – Java Database Connectivity

JDRBAC – Java Database Role-based Access Control

JSSE – Java Secure Sockets Extension

NIST – National Institute of Standards and Technology

RBAC – Role-based Access Control

RMI – Remote Method Invocation

SQL – Structured Query Language

SSD – Static Separation of Duty

SSL – Secure Sockets Layer

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

The authors wish to gratefully acknowledge the guidance, wisdom, and patience of their thesis advisors, Dr. James Bret Michael and Dr. John Osmundson. Faouzi Hammoudi dedicates his work to his parents Mekki and Cherifa Hammoudi, his wife Habiba, and daughter Chaima for the time taken from them, for their encouragement, and for their never-ending patience and love.

THIS PAGE INTENTIONALLY LEFT BLANK

I. BACKGROUND

A. INTRODUCTION

Much of the work to date to apply Role-Based Access Control (RBAC) to database management systems has focused on single database systems or an integrated distributed database system. In the military, situations such as joint operations arise where the necessary data may be stored on databases that exist in separate organizations. Such organizations may consist of different branches of the armed forces, distinct government agencies, or various allied countries. A single composite database may be too time-consuming or costly to implement. Direct integration of the databases may be impractical due to security concerns as well as cost issues.

For situations where the need exists to consolidate multiple independent databases, and where the direct integration of the databases is neither practical nor desirable, the application of RBAC requires that policy be enforced via a method that is distinct from the databases. The method must provide for the verification of the RBAC policy, while allowing for the independence of the various databases on which the policy is enforced.

This thesis proposes an application-level model that provides for a web-based interface for users to be granted access to data held in various independent databases. The solution enforces a strict RBAC policy on a well-defined set of accesses, while alleviating the need for users to have a separate account on each of the databases. The specifications of the model are detailed, and a prototype application is described that provides the necessary functionality. Critical issues such as the security of the application and its communication channels are discussed, and areas requiring future research are also noted.

B. DISCUSSION

Role-based access control (RBAC) has been used in computer systems for at least twenty years, but only within the past few years have rigorously defined general-purpose RBAC models and implementations begun to appear. RBAC is a continuously evolving, rich, and open-ended technology. For instance, various RBAC models are now being

embedded in commercial-off-the-shelf software-based products, such as database management and operating systems. Moreover, RBAC has been specialized for use with component-based message-passing architectures. The U.S. National Institute of Standards and Technology (NIST) has proposed a standard reference model to facilitate interoperability among information systems that implement RBAC. NIST views RBAC as a tool to enable the administration of security at a business-enterprise level rather at the user-identity level.

A role is a semantic construct forming the basis of access control policy. Roles can be assigned according to, for example, job functions performed in an enterprise. An enterprise can be defined, for the purposes of this thesis, as any large organization that utilizes distributed computing systems to support its operations. Once a role class is defined, permissions and authorizations can be assigned to that role. Among the benefits to an enterprise of using roles are the ability to administratively define and enforce enterprise-specific security policies that cannot be achieved using other methods of access control, and to dramatically streamline the process of authorization management. For enterprise security, the importance of a role lies in its persistence within the enterprise-computing paradigm. The permissions associated with a role can change as the functions within an enterprise evolve over time. Membership within a role may be highly transient as new employees join, and new job positions and assignments are created. However, the notion of a role remains a relative constant. For example, a ship may have constant turnover among commanding officers, and the duties of a ship's captain may evolve slowly over time, but the notion of the ship's captain will remain a relative constant. This persistence is what makes roles so attractive for managing permissions and authorizations. RBAC makes it possible to separate the high-level policy management from the low-level details required to assign permissions for operations directly to individual users.

In addition roles being persistent, enterprise access control policies for authorizing users are also natural and consistent with roles. Roles can be used to specify competency, responsibility, authority, and delegation for tasks. Coupled with the notion of constraints, roles can be used to enforce conflict-of-interest (e.g., separation-of-duty) and workflow policies, among others.

C. NIST RBAC MODEL

The NIST RBAC model is organized as four categories of abstractions of increasing complexity. The levels are cumulative in that a higher level abstraction subsumes the expressiveness of the lower level abstraction. The four categories and the rationale for each one are given below.

1. Flat RBAC

The flat RBAC abstraction is the basis for the entire model, and is the most primitive of the four levels of abstraction. The relationship between users, roles, and permissions constitutes the base of the model. The definitions of the entities and relationships in the model are as follows:

User: A user in this model is a human being or other autonomous agent such as a process or a computer.

Role: A role is a job function or job title within the enterprise with some associated semantics regarding the authority and responsibility conferred on a member of a particular role.

Permission: A permission is an approval of a particular mode of access to one or more objects in the system. Permissions confer the ability of the holder to perform some action or actions in the system. In addition, each permission can be represented as either a permission or prohibition.

User-Role: This relationship represents which user is assigned to perform what kind of role in the enterprise.

Permission-Role: Assigns permission or a set of permissions to a specific role or a set of roles.

Role-Role: Specifies the hierarchy between roles.

Users are assigned to roles, permissions are assigned to roles, and users acquire permissions by being assigned roles. The relationships user-role and role-permissions can be many-to-many (Figure 1).

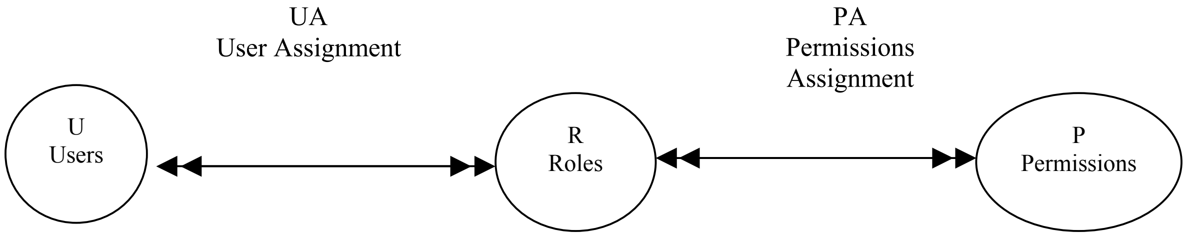

Figure 1. Flat RBAC

The flat RBAC abstraction supports user-role review, thus providing a basis for the determination of all users that are granted a specific role and the roles that are permitted for a specific user. Moreover, users can simultaneously exercise permissions of multiple roles, akin to multiple inheritances in an object-oriented model.

The rationale for the flat RBAC abstraction is based on the notion that conventional group-based access control is robust. Although the NIST model does not require support for sessions with discretionary role activation, it does require the ability to activate multiple roles simultaneously and in single sessions.

The requirement for user-role review differentiates the flat RBAC abstraction from the group-based access control modeling paradigm. However, the flat RBAC abstraction leaves many issues open related to the scalability of the model, the nature of the permissions, expression of permission revocation, and representation of role administration.

2. Hierarchical RBAC

The hierarchical model extends the flat RBAC abstraction by introducing the notion of role hierarchies (see figure 2). A hierarchy is a partial ordering of roles, whereby senior roles subsume the permissions of their juniors. Role hierarchies are a natural means for structuring roles to reflect an enterprise's line of authority and responsibility. They can be inheritance hierarchies, meaning that the activation of an instance of a senior role by a user (such as at login) implies the inheritance of the permissions of all junior roles (see figure 2a), or activation hierarchies, in which there is no implication of overall inheritance of permissions (see figure 2b). In the activation hierarchy, the inheritance is limited to the roles which are subordinate to the specified role in the tree structure of the model. Therefore, the NIST model has identified two sub-

levels, which are the general (inheritance) hierarchical RBAC that uses the partial ordering of roles, and the restricted (activation) hierarchical RBAC that uses simple structures such as trees or inverted trees.

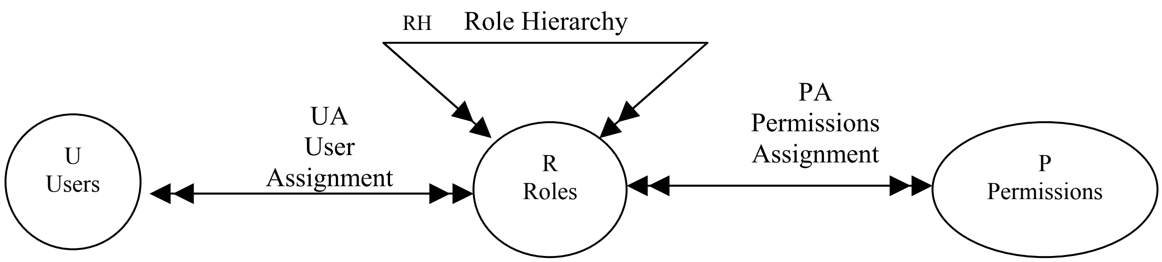

Figure 2. Hierarchical RBAC

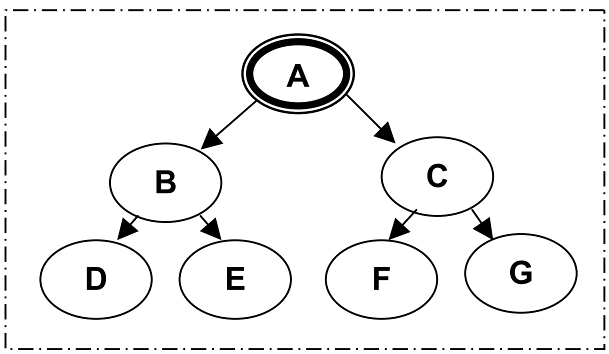

Figure 2a. Inheritance Hierarchy

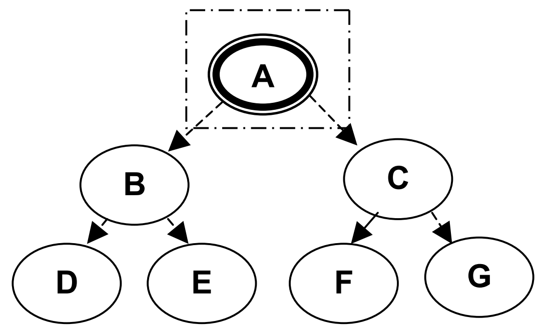

Figure 2b. Activation Hierarchy

The limited degree of inheritance afforded by this abstraction supports control over power aggregation for a given role, and provides a way to prevent user errors or malicious software from adversely affecting the system.

3. CONSTRAINED RBAC

The constrained RBAC model introduces the semantics needed to enforce separation of duty (SOD), which is a time-honored technique for mitigating the potential for the occurrence of fraud and accidental damage attributed to sharing of duties. It is often used to enforce conflict-of-interest policy that enterprises may employ to prevent users from exceeding a reasonable level of authority for their role. The semantics of the model at this level encompasses the principle of least privilege: users are given no more

than the necessary privileges to perform their roles. The two categories of SOD are as follows:

a. Static Separation of Duty (SSD)

A constraint associated with the user-role assignment (see figure 3). SSD is a way to enforce mutual exclusion between roles and can be centrally specified and uniformly imposed on specific roles.

b. Dynamic Separation of Duty (DSD)

A constraint associated with the activation of roles within user sessions (see figure 4). It provides an organization with the capability to address potential conflicts of interest at the time a user's membership is authorized for a role.

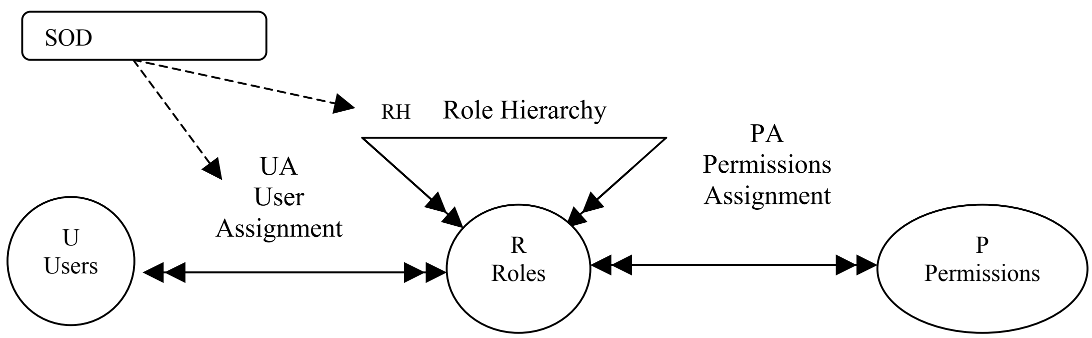

Figure 3. Constrained RBAC- Static SOD

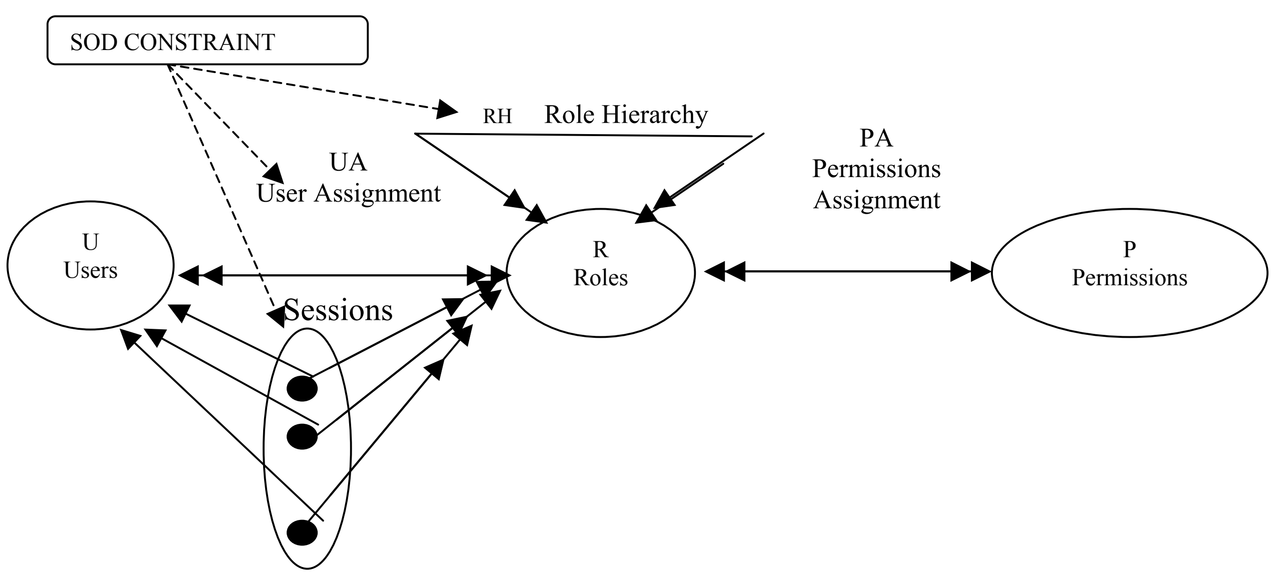

Figure 4. Constrained RBAC- dynamic SOD

4. SYMMETRIC RBAC

Symmetric RBAC extends the semantics of the model to accommodate permission-role review, which is similar to user-role review: it is possible to determine the role to which a particular permission is assigned as well as the permission assigned to a specific role. The permission-role review interface returns one of the two types of results. The query-symmetric RBAC will include the semantics necessary for defining direct and indirect assignment of permissions. Direct-permission assignment pertains to the set of permissions that are assigned to the user directly. Indirect permission assignment includes the direct permissions assignment and the set of permissions that are inherited by the roles assigned to the user.

D. OBSERVATIONS

The NIST RBAC model provides a simple and methodic way to understand and consequently implement enterprise security policy for user access control in a manner that reduces the administrative overhead from the management of the access control policy for the enterprise. Today, the existing implementations of RBAC are of two categories. In the first category, we have implementations that are part of the operating system, representing a module that mitigates the access control to the enterprise system. The second category is an implementation that is part of the database management system, where all access controls are mediated through that layer. The two precedent solutions are somewhat problematic because of the potential compatibility and interoperability issues that arise in a large-scale, distributed computing environment. The solution that will be proposed in this thesis is to implement a general purpose mechanism that is platform independent and that will allow the negotiation of the user access to distributed databases across the boundaries of various enterprises. The mechanism will rise above the level of a single organization, to function across any number of distinct entities without dependence on any one enterprise.

THIS PAGE INTENTIONALLY LEFT BLANK

II. APPROACH

A. DATABASE ROLE-BASED ACCESS CONTROL (DRBAC)

Database RBAC is similar in many ways to the RBAC policy that would be implemented in an operating system. Both are based on common tasks, with users assigned to roles that are designed to provide sufficient access to the information required to perform the tasks. The role is defined as a subset of privileges to specific units of data, and a subset of operations that can be performed on these units. The user's access to data is controlled via the roles to which they are assigned, rather than through the specific units of data. This higher level of control provides a higher degree of flexibility in administration of privileges.

While an operating system policy is directed towards controlling access to particular files and directories, a database policy is focused on the queries that enable access to the information in the database. So the RBAC implementation in a database becomes, in general terms, query validation. The access control is done by validating the user query, based on the access privileges associated with the user's role.

B. QUERY VALIDATION

The policy for query validation can be thought of in similar terms to the policy for firewalls. A firewall policy can be stated in one of two ways: 1) allow access to everything unless there is a specific rule to deny access, or 2) deny access to everything unless there is a specific rule to allow access. The second method is generally accepted as the more secure approach. For query validation, in this thesis we adopt the latter way of interpreting policy: access to the database will not be granted unless the query is specifically validated.

A query can be broken down into four parts: the tables of the database that are to be accessed, the operation that will be performed, the specific tablefields that contain the data of interest, and any conditions that apply. A basic format for a query in the relational database model, expressed in the Structured Query Language (SQL), is as follows:

Operation Tablefield [*Tablefield*, ...]

FROM *Table* [*Table*, ...]

WHERE *Condition*

[AND] [*Condition*] ...

We have a given operation on one or more tablefields, which are taken from one or more tables, with one or more conditions applied to the query. In validating the query, the RBAC policy would have to verify that the user has permission to access data held within the tables and the specific tablefields, and that the user can perform the given operation on the tables within the boundaries established by the condition or conditions provided. Hence, we define a role as a combination or combinations of the four elements of a query, from which any query that a user wishes to execute on the database can be compared to and validated against the authorizations and permissions of that role.

C. WHY JAVA?

In a distributed computing environment, information sharing can be critical to the successful operation of an organization. Information needs to be provided to remote users who may be using a variety of operating systems, platforms, web browsers, etc. Java provides a means for development of an application that can be accessed by remote users regardless of the platform or software employed by the user. The user can utilize a web browser to access the server running the application, and multiple users can be given access simultaneously. The application can be tailored for different users, and reconfigured to provide for changing requirements.

For our Java Database Role-Based Access Control (JDRBAC) application, the same code that creates the user interface also provides the RBAC policy implementation. This allows for independence of the application from the databases that it accesses. The application can establish a connection to the database, along with a defined scope of access within the DBMS. The application can then provide subsets of its access to users in the form of roles. The users do not need an account on the DBMS, because they log into a role defined by the application. If the user's needs change, the application can transfer the user into a new role, or tailor the user's existing role.

The application runs separately from the DBMS, thus affording flexibility within the architectural framework. The JDRBAC application can reside on the same server as the database, or be located on a separate web server (see figures 5a and 5b). Separation of the application and the DBMS can allow for additional layers of security to be in place between the application server and the database system, such as a proxy server or a firewall.

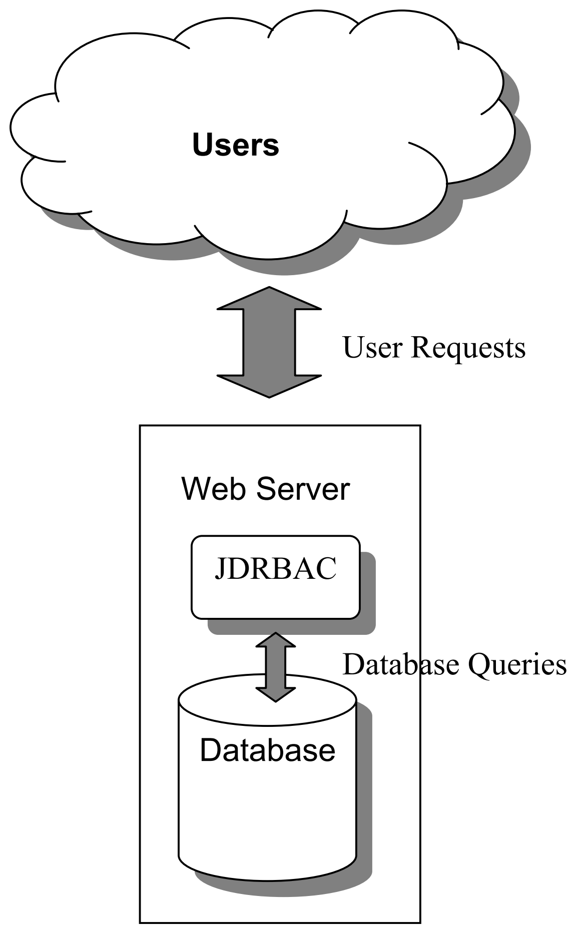

Figure 5a. JDRBAC co-located on Database Server

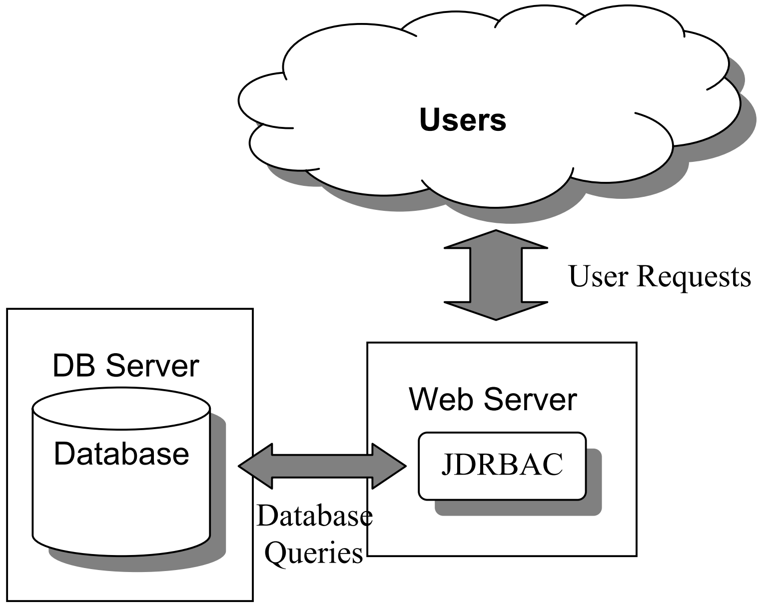

Figure 5b. JDRBAC on a separate web server

The application can provide access to a distributed database, (see figures 6 and 7).

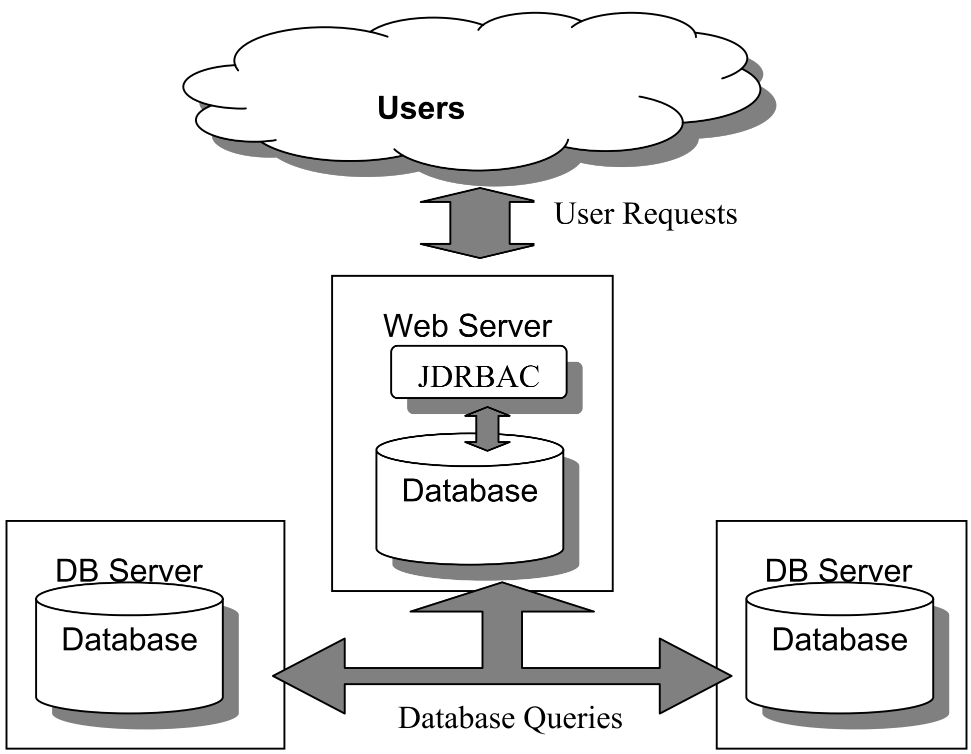

Figure 6. JDRBAC access to distributed DBMSs through co-location with single database

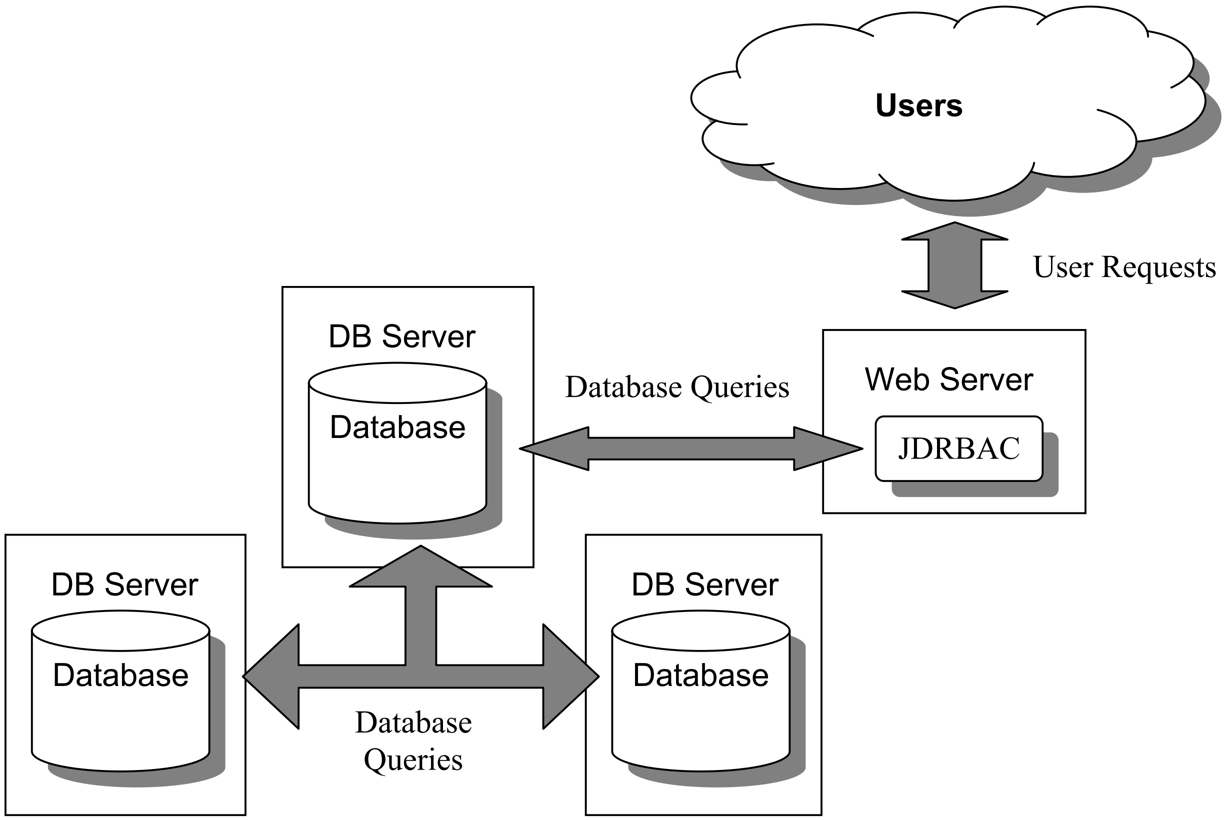

Figure 7. JDRBAC access to distributed DBMSs with location distinct from databases

The primary advantage of implementing the RBAC policy through the application is that it allows for the application to access multiple databases, across various platforms, operating systems, or flavors of database management systems. The application can potentially allow a user access to multiple databases, without having an account on any of them (see figures 8 and 9). The access control policies of the databases can be configured separately from the access control policy of the application. This simplifies management of the databases, and centralizes control of the users' access.

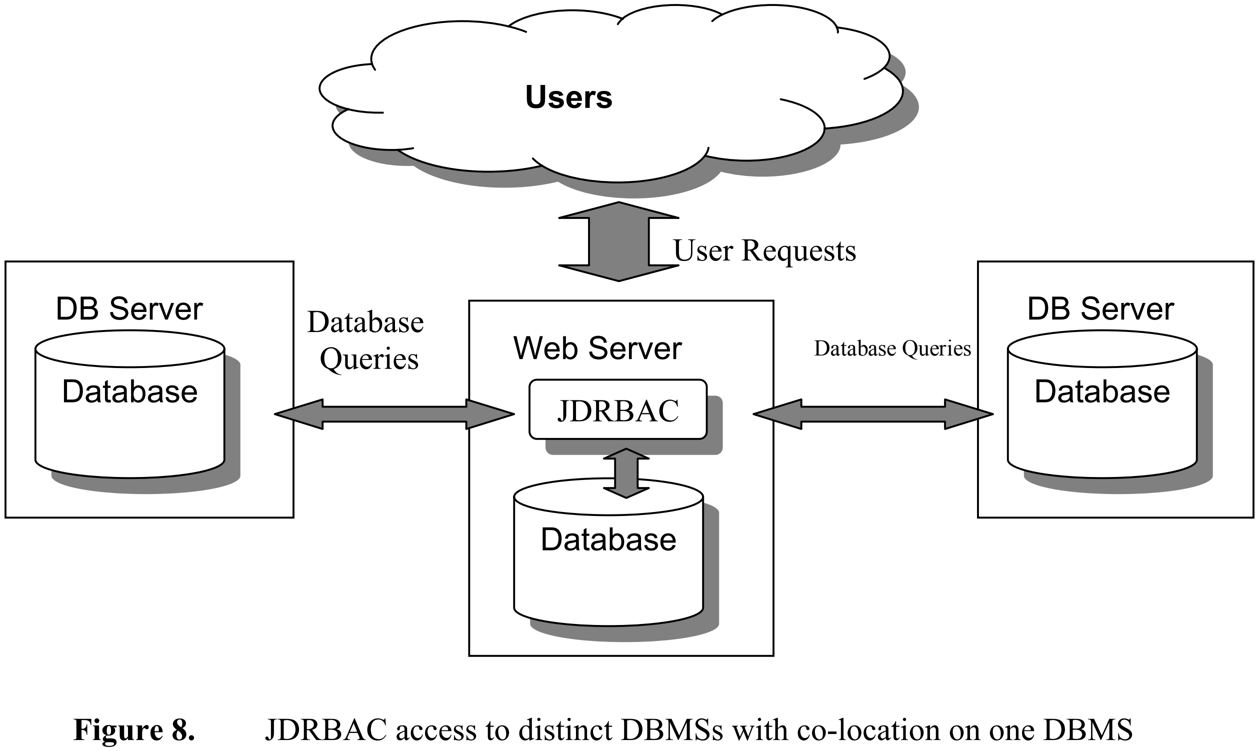

Figure 8. JDRBAC access to distinct DBMSs with co-location on one DBMS

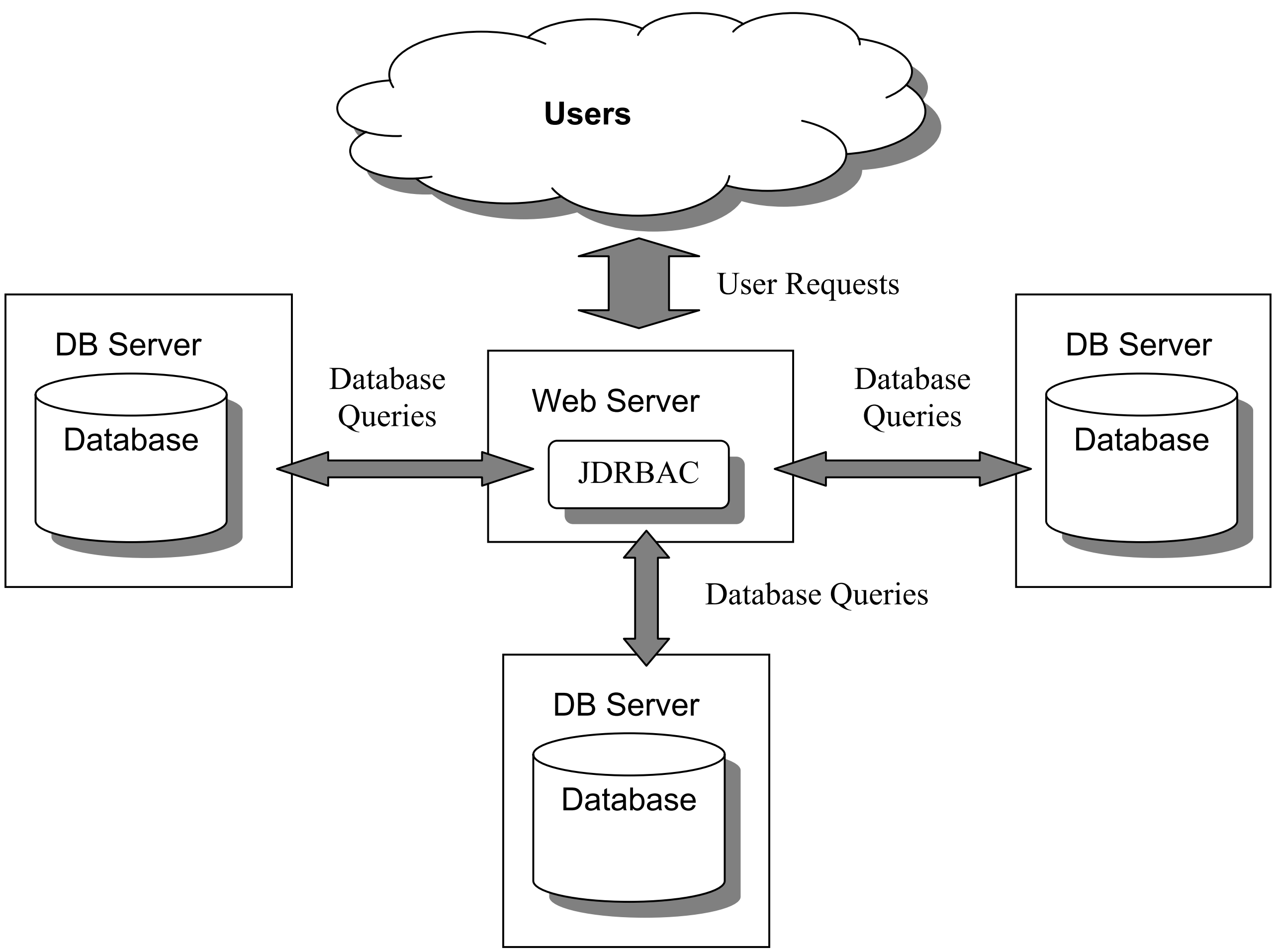

Figure 9. JDRBAC access to distributed DBMSs through separate location on web server

D. TREE STRUCTURE

The JDRBAC implementation could be realized using tables within a database. Tables could be defined within the database that contains the roles, with links to other tables that provide for the various accesses that are given to the roles. But this directly ties the application to the particular database. Alternatively, the application could take advantage of the role mechanisms included with newer databases, such as Oracle 8i and 9i. Once again, this ties the application to the database, and does not allow for easy integration of other platforms (such as Microsoft SQL server).

By providing a distinct data structure for the application, the platform independence is maintained, as well as the ability to integrate access to multiple databases through the application.

A tree structure can provide an efficient and effective means of implementing the query validation mechanism of the JDRBAC. Within the tree, the roles would be defined as the children of the root node. Under each role, the first set of children would be the databases to which the role has been granted access. The children of each database node would be the tables of that database that are available to the user. The children of each table would be the tablefields that the user can view. The tablefield children would be the operations (read (SELECT), write (INSERT), edit, etc.) that can be performed on that tablefield. And the children of the operations, the leaves of the tree, would be any conditional statements (e.g., username = "Smith") that restrict the operation on that tablefield.

This structure would allow for a top-down approach to the query validation. The initial check of the query would be to determine the role of the user, which determines the first child of the root (i.e., the role). Then the databases to be accessed by the query would be checked against the children of the role node. From the databases, the tables within each database would be checked for granted access by the role. If all tables exist as children, the query validation continues. If not, then the query is invalid the operation stops.

The next step is to check whether a child or children of the table node exist for the tablefields that the user wishes to view or change. If the tablefields are present within the

role, then the desired operation (e.g., read) is checked for permission. If the operation is not present, then the query is invalid, and the operation stops. Finally, any defined conditions for the tablefield are checked to ensure that the required restrictions are included in the query.

In the described process, an invalid query is blocked in the shortest number of steps, while a valid query must be checked all the way down to the leaves. While this may seem unwieldy, it provides the necessary method to invalidate all queries unless specifically allowed.

In a large database with many tables, the tree could become very large if all tables and fields are included as children. To preclude this, an 'ALL' child could be included within the tree structure. Then if a role has read access on all the tables of the database, the database node would have an ALL child that would represent all of the tables of the database, and the table node would have a subsequent ALL child at the next level to represent all of the tablefields of the table. This ALL child would have the read operation as its child. This would allow for shorter verification searches for roles that have been given widespread privileges.

E. PROCESS

The overall process for the JDRBAC application can be broken down into the following steps (see figure 10):

1. User Creation of Query

The first step will be a user interface that will enable the development of the query from the user input. The user can select data fields, and what operation to perform on those fields. The application can provide all of the possible fields for the user to select from, or limit the display to those fields that are available to the user via the role of that user.

2. Query Validation

At this point the RBAC policy implementation will take place. The query will be validated via the step-by-step approach detailed previously, and an invalid query will produce an indication to the user via the interface.

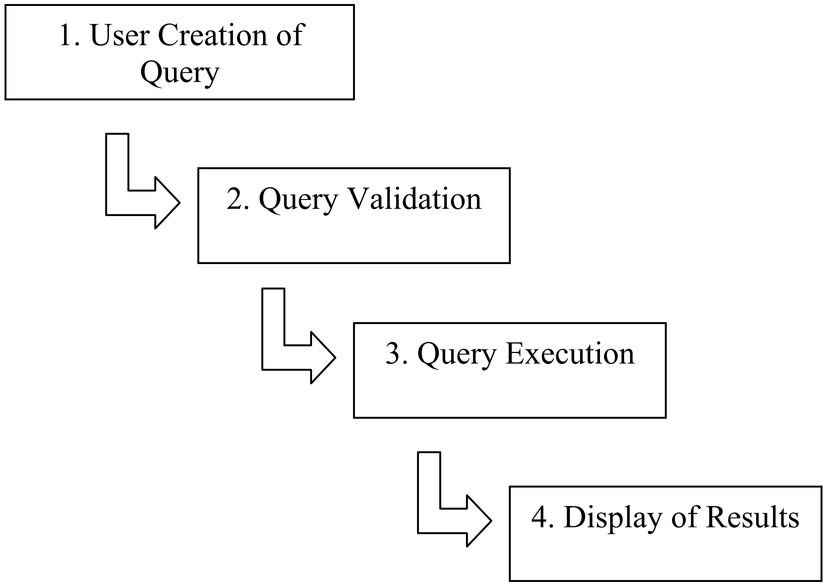

Figure 10. JDRBAC process

3. Query Execution

The application will access the required database or databases for the requested datafields. The database access will be done via secure channels, and the results will be combined as necessary into a single comprehensive result.

4. Display of Results

The result of the user query will be presented on the user's web page in an appropriate manner.

In this thesis we describe a prototype that we developed to implement the preceding list of operations, with the necessary web/database application. Our prototype of the JDRBAC is based on the Java-based RBAC implementation developed by NIST, adapted to the tree structure described in Chapter III. Our intent is to create an effective and flexible application that can be easily fielded in any situation, providing necessary access controls, user functionality, and levels of security in its operation.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN

A. REQUIREMENTS ANALYSIS

The JDRBAC implementation, as previously discussed, is to be used in a distributed environment. The application needs to support the following transparencies: access, location, concurrency, and mobility. These transparencies allow users to pull information from one or more databases that could be developed on different database management system technologies. In this thesis, the application architecture comprises two high-level sets of modules (see figure 11), each of which will be examined in more detail later. The first set is an RBAC policy implementation that will function similar to a reference-validation mechanism. It will be consistently invoked to validate the users' actions against their roles and permissions. The second set of modules will be the web-application of the functionality supporting the access and use of the distributed databases. This application will use the RBAC policy implementation to validate the user queries in a step-by-step fashion. The communications between the two modules are of two fundamental types. The first type, explicit and user-driven, are method calls for the purpose of checking the user action against policy and returning the result. The second type, implicit or event-driven, are method calls that constantly check the validity and consistency between the user actions and the user's roles and permissions in the time domain. These can be local or remote method invocations.

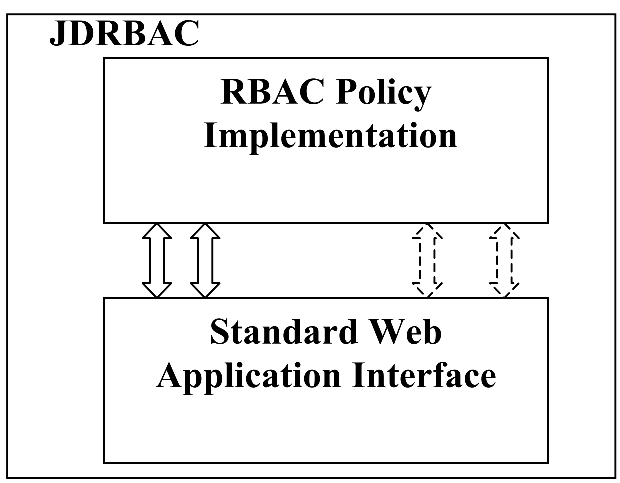

Figure 11. JDRBAC Architecture

The web application under consideration has to meet several requirements, which include the following:

1. Security

The web application provides role-based access control security that is defined by relationships among users, represented in the JDRBAC tree. Moreover, the application will provide a user identification and authentication module that will be used in the login process. Additional security, such as a firewall, an application gateway, encryption or secure sockets layer, can be used to provide defense in depth for the entire application environment. The implementation of such security mechanisms is beyond the scope of this thesis.

2. Platform Independence

One of the fundamental motivations for the implementation of JDRBAC is platform independence. Among the primary problems encountered by enterprises is the integration of their systems in a distributed environment, and satisfying the need for a global context in the information systems. The enterprise is typically in possession of legacy systems and new systems that utilize different operating systems and a variety of database management systems. Therefore, there is a need for applications to accommodate such heterogeneity, and be able to interface to the existing databases of various enterprises.

3. Access Transparency

The manipulation of multiple enterprise data requires skilled operators and personnel who know how to query the databases and how to store relevant information. It is typically essential for these personnel to be proficient in some data manipulation language (e.g., SQL) in order to query or update the databases. The JDRBAC implementation will provide access transparency so that users can manipulate the data from remote databases without the specific knowledge of any querying language: they will use a standard interface.

4. Flexibility

The JDRBAC application will consist of modules that have strong coherence and weak cohesion. The modular approach allows for having an adaptable architecture that

can be readily tailored and tested to address the evolving requirements and missions of the various enterprises involved. Application portability allows for the consolidation and integration of proprietary systems in a computer-support for collaborative work (CSCW) environment.

5. Ease of Management

The neutrality of the RBAC policy stems from its ability to be applied to any given enterprise environment. Its nature allows for the definition of generic user-role and role-permissions relationships that, in turn, can be used in the management of authorizations. This is especially desirable for enterprises intending to consolidate their efforts in a CSCW environment.

6. Cost Effective

The implementation of the RBAC policy, in theory, could minimize the overhead associated with maintaining an access-control policy.

B. USER INTERACTION PROCESS

The JDRBAC application will interface with two types of users, the common user who accesses data, and the administrative user who maintains the application. While the common users will have their access controlled by the JDRBAC tree, the administrative user can view and modify the entire JDRBAC tree.

In order to update the JDRBAC tree, the administrative user will be provided an interface to view the entire tree structure, from which the user can select specific nodes to perform operations on. In this sense, the interface will be similar to the one provided to the common user, with the significant addition of being able to view the entire database of objects that are provided to the application. In the case of adding database objects to the tree, there will need to be a separate channel for the administrator to obtain information about such objects from an authorized database administrator. This is because the JDRBAC application itself will need to be provided access to such objects via a database administrator before they can be added to the JDRBAC tree.

In the case of the common user, the primary function will be to view data on a read-only basis. The ability to add or delete objects from the database will not be provided to a JDRBAC user. In order to achieve the primary goal of eliminating the need

for every user to have an account on every database accessed, only the JDRBAC application will be provided an account, and will provide partitioning of its access to the application users. So application users are essentially anonymous to the databases, and it is undesirable to provide anonymous users with the ability to directly affect the structure of a database. Such operations will be reserved for database users who are specifically authorized to perform such actions, that is, "trusted users."

The ability of common users to edit data held within the database provides advantages, and presents problems. Most implementations of JDRBAC will likely be created to provide only the read access to data. Allowing users to edit existing data raises the issue of concurrency control. If two or more users are accessing the same data at the same time, and all have the ability to edit that data, then which changes should take precedence when the users commit them is vital to maintaining a consistent database state. The problem is made even more complex with the addition of users who are accessing the data directly as authorized database users, the "trusted users" previously noted. Maintaining currency transparency (i.e., hiding multiple accesses to shared resources from the users) is possible through mechanisms introduced in the database, since the JDRBAC application is seen as simply an authorized user accessing the database. However, the inclusion of the means to provide such transparency, via such things as transaction controls, to the users of the JDRBAC application is beyond the scope of this thesis. Such cases can be reduced by limiting edit (i.e., the changing of existing data) or write (i.e., the entering of data into a previously empty field) operations on specific data to only one specific user (e.g., in the case of a user's own personal data), but tailoring access to a specific user runs contrary to the notion of RBAC policy.

But some implementations, such as those providing access to personnel databases, or user account information, could be set up to allow users to access and edit their personal data. In order to address such cases, the JDRBAC application incorporates the write (i.e., entering data into a previously empty field), and edit (i.e., changing data currently existing in a tablefield) operations into the design and implementation description of the JDRBAC application. Issues such as concurrency control are left to future research.

Within the parameters of the allowed operations, the step-by-step process of common user interaction with the application is defined in the following steps:

1. Login – Authentication (JDRBAC policy implementation)
   The user enters the appropriate login identification and password, which is verified via a module contained within the RBAC policy implementation. The table will hold the user identification, the password for that user, and the role or roles that have been assigned to the user. The login process is necessary to ensure that users are associated with the proper roles.

2. Present possible roles (JDRBAC tree)
   Once the user has been authenticated and associated with a role or roles, the roles will be displayed to allow for user selection. Because roles can have different accesses associated with them, distinct roles may conflict with one another in terms of permissions. This is due to the activation structure of the RBAC policy, rather than a hierarchical structure.

3. User selects roles
   The user will select the role or roles based on the specific data that is to be viewed or edited.
4. Present all databases available for roles (JDRBAC tree)
   The application will search the JDRBAC tree structure to determine the databases associated with the roles. These databases are the children of the role nodes within the JDRBAC tree structure.
5. User selects databases
   The user will select from the displayed databases to access the desired data.
6. Present tables from selected databases (JDRBAC tree)
   The application will search the JDRBAC tree (specifically, the children of the database nodes) to obtain the tables associated with the selected databases.

7. User selects tables for view formulation
   The user will select the specific tables for which access is desired.
8. Present tablefields from selected tables (JDRBAC tree)

The application will search the JDRBAC tree (specifically, the children of the table nodes) to obtain the tablefields within those tables for which the user's role has been granted access.

9. User selects tablefields for the view
   The user will select the specific tablefields that contain the desired data.

10. User selects operation to be performed on data (read, write, edit) (JDRBAC tree)

    The user selects the desired operation for those displayed. The operations displayed will be taken from the JDRBAC tree, as the operations allowed on the tablefields for the role or roles of the user. The restrictions on available operations have been previously discussed, as well as the potential issues associated with the edit and write operations.

11. Application accesses databases for data (Query Builder/Query Execution)
    At this point, the application can create an SQL-form query for the user selections. The application can then execute the query on the appropriate databases to retrieve the necessary data.

12. If read operation is selected, then the application presents data to user
    If the user has selected to read the data, the results of the query are presented in an appropriate format for the user to view.

13(a). If write or edit operation selected, form presented for user to write/edit data
If the user has chosen to edit data or enter new data, an appropriate form is displayed for the user to view the existing data (if any). The user can then fill in any blank fields in the records (write) or change existing data in the records (edit).

13(b). Application commits data to database
If changes have been made to the data presented to the user via an edit or write operation, then a SQL form query will be created to enter the new data into the database, and commit the required changes.

C. JDRBAC TREE DESIGN

The RBAC policy implementation will consist primarily of three modules: one for interaction with the JDRBAC tree, one for user validation, and one for accessing the

associated database. An additional module may be needed for aliasing. Aliasing would be required to provide user-friendly descriptions of otherwise obtusely-named database objects. Translation of interface descriptions to actual object names will need to be done to enable the construction of well-formed queries.

The user validation module will simply provide a means to validate the user identification and password at login, and associate the user with the role or roles to which they are assigned. The implementation of this module can be done with a simple table containing the user identification, password, and all roles that the user has been assigned.

The JDRBAC tree will be implemented in a module that will require two distinct interfaces. One will be for user operations, where the application will access the tree to search for the objects to which the user has been granted access. The second interface will be for administration, to allow for an authorized user to update, create, or delete roles from the tree.

The design of the tree will focus on user operations, as the likelihood is that multiple users will access the application simultaneously. There is a need to make search operations efficient during user operations, in order to provide necessary levels of speed in presenting data. The interface can be adapted to provide the necessary views for the administrative users.

Each level of the tree consists of a different level of access provided to the role, and thereby to the user. The number of children of each node of the tree can vary. For instance, there can be any number of roles defined, a role can have access to a number of databases, and a database can consist of several tables.

For a user process, it is necessary to access and present all children of a given node to the user. When the user selects a given node or nodes (be they databases, tables, tablefields, etc.), the operation is repeated. So the optimal design of the tree structure should allow for ease of searching on a particular level of the tree, specifically the children of a given node.

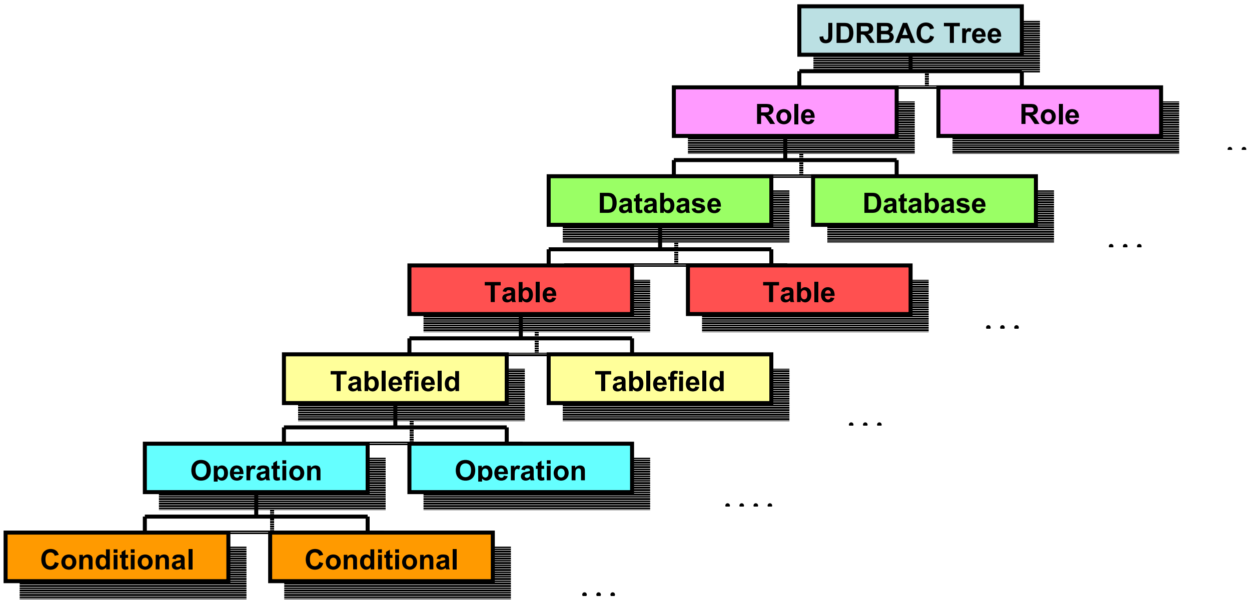

Figure 12. JDRBAC Tree

In this regard, the children of a particular node can be considered peers of one another. The databases that can be accessed by a specific role are peers, and the tables of a particular database of that role are also peers. Thus each element of the tree must be able to reference both its children and its peers on the same level.

An effective means of representing the JDRBAC tree is in the use of linked lists, with each node of the tree containing the necessary data, a child pointer, and a peer pointer:

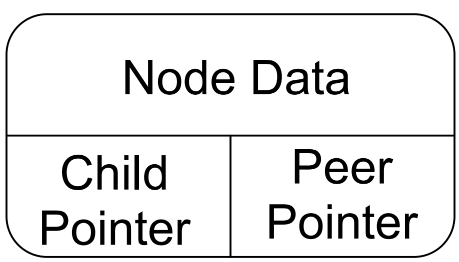

Figure 13. JDRBAC Tree node

The node data would provide the type of object that it represents (e.g., database, table, tablefield), the descriptive name of the object, and any other required items. The pointer would be set to null, or to the associated child or peer node. The root node would have a peer pointer set to null, and its child pointer set to point to the first child. The first child would point to the next child with its peer pointer, with the rest of the children linked in the same way. Each node would have its child pointer pointing to its children, and so on. An example of the resulting tree, for the databases, tables, and tablefields of a given role, would be as shown in figure 14.

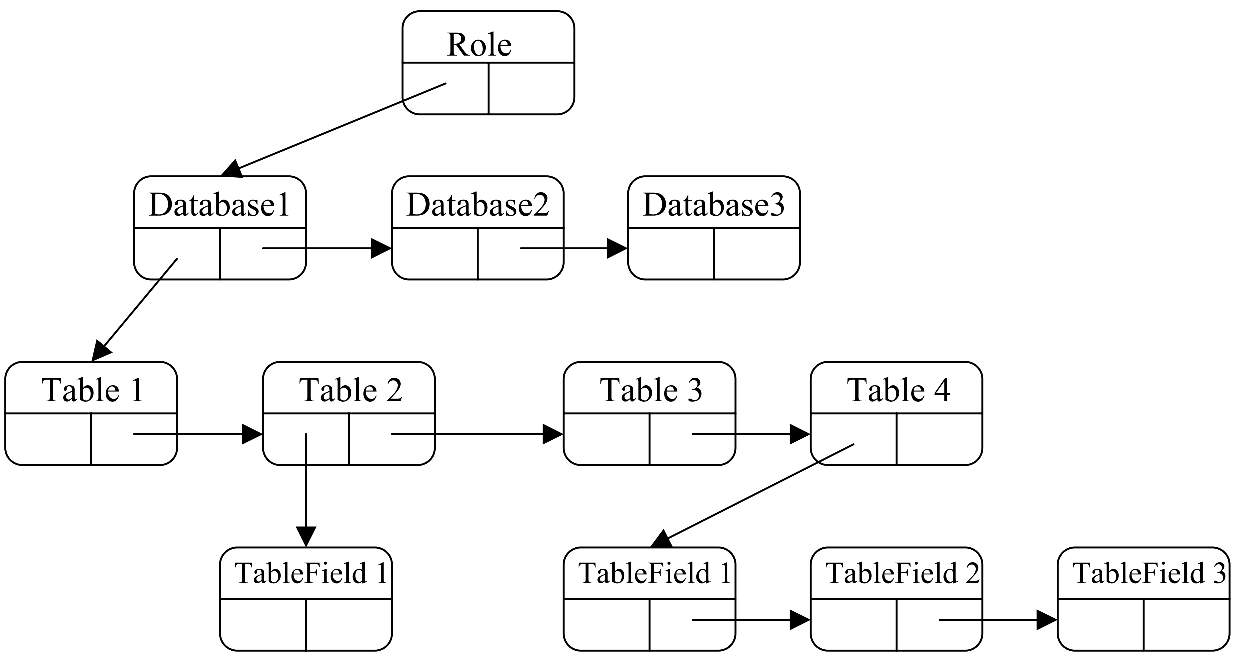

Figure 14. Example of a JDRBAC Tree

D. APPLICATION DESIGN

To satisfy the needs of the application and the requirements stated in the requirements analysis phase, the application can be composed of four modules (see figure

15). The correct use of the application depends on how each module is developed, and on how well the modules are integrated and communicate together. The four modules are given below.

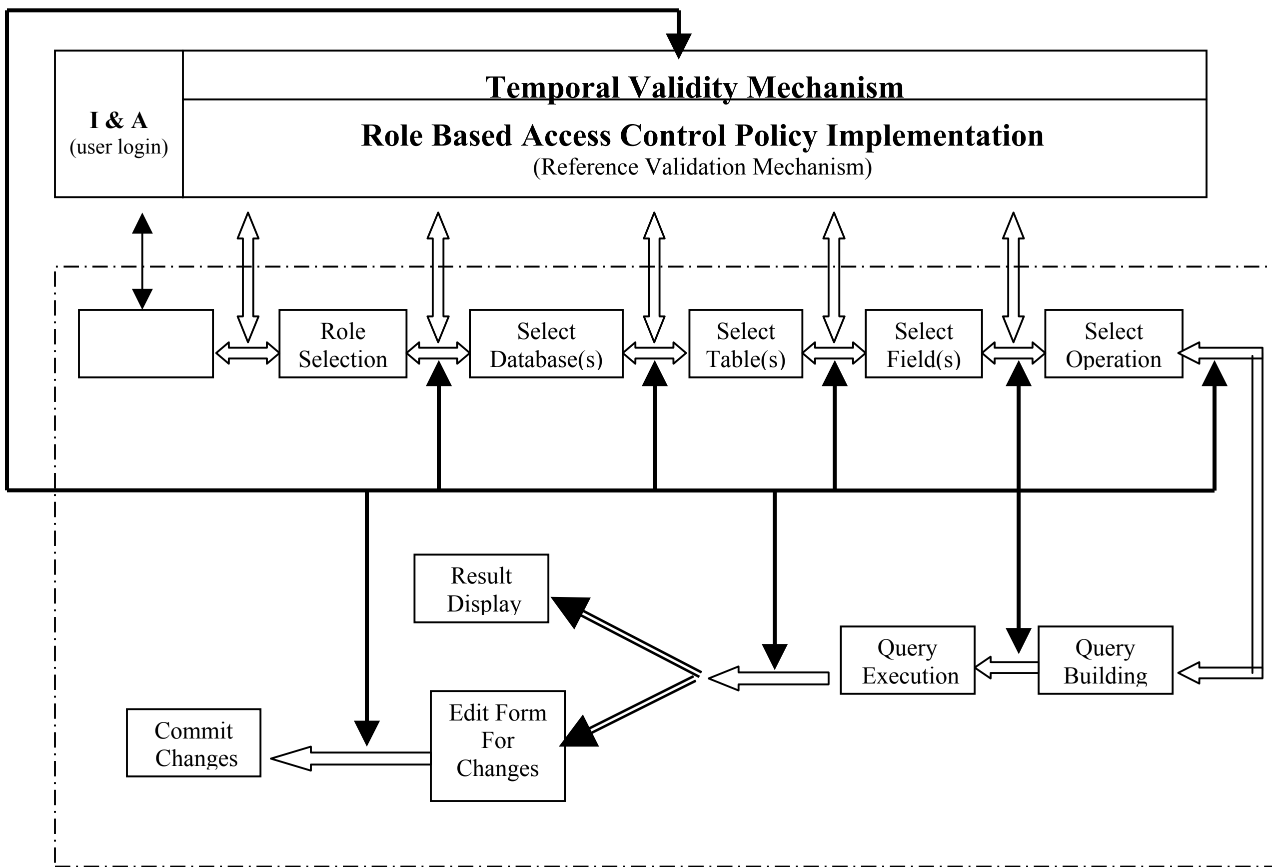

Figure 15. JDRBAC application architecture

1. Identification and Authentication

The identification and authentication module can be used to enforce the security policy of the application. The users have to supply the correct credentials to be able to utilize the system. They will be provided a maximum of three attempts to log into the application; otherwise the system will lock out the user from the application until the administrator re-establishes access for that user. The identification and authentication module will enforce proper standards of protection for the usernames and passwords, in terms of password alphanumeric composition and length, unique user identifications, and time limits for account validity.

2. Role based access control policy implementation

This module represents the implementation of the RBAC policy and controls the delivery of the information to authorized users. It is in the kernel of the JDRBAC application where the actions of the users are validated against the active role or set of roles selected when logging into the system.

The application presents only the information the users are allowed to see. The enforcement of the RBAC policy applies down to the level of granularity specified by the lowest level of the JDRBAC tree structure. Therefore, on every step of the user process the system provides only the data to which the user is granted permission; this will prevent unauthorized disclosure of information. The design of this module consists of the implementation of a tree structure that defines the user-roles and role-permissions relationships, and the hierarchies between the roles. The model accounts for dynamic changes in the authorization state and constantly re-evaluates the user's role or roles to ensure a consistent state of currently authorized access during the user process. The bi-directional arrows between the web application and the role-based access control policy implementation (see figure 15) indicate that a user request (for data) is forwarded from the web application to the RBAC implementation, which processes it, in conformance with the set of rules and permissions that are active at that time, and sends back the results (i.e., the desired data) to the user.

3. Temporal Validation Mechanism

Recently, Bertino, Bonatti and Ferrari introduced Temporal Role Based Access Control (TRBAC). The basic idea of the TRBAC relies in the fact that roles can be active in certain periods and non-active at others. Moreover, there can be activation dependencies among roles. Roles and relative permissions can also be revoked dynamically, while the users are actively involved in a session. Therefore, the system has to take appropriate action to dynamically isolate the user from the roles or part of the roles in the most appropriate manner and in a way to prevent unintentional corruption or unauthorized disclosure of the data in a case of a change in the user role assignment.

In theory, the temporal validation mechanism has to detect any change in the initial set of user-role-permissions assignments and trigger the correct actions. If the

temporal validation mechanism detects a change in the RBAC policy that affects the current access restrictions of the user process, it triggers the system to examine all of the previous steps in the process, return to a consistent state or disconnect the user from its role in case of role revocation. The design and implementation of the temporal validation mechanism is beyond the scope of this thesis and is left for future work.

4. JDRBAC Web application

The JDRBAC web application is a standard application that is platform independent and adaptable. The application is intended to operate in a distributed environment and to provide access to heterogeneous databases.

The connection to the various databases is kept transparent to the user and is mediated through the use of a database connectivity function that is supported by most of today’s database management systems. Moreover, the users are not required to have different accounts on every database management system to be able to access the data. Rather, it is sufficient that the web application has only one account on each system, and the users get the access though the roles that are assigned to them. This demonstrates the potential benefits of the application of the RBAC policy on a distributed environment in the management of the user accounts and the relative permissions.

The web application is composed of sub models that implement the user process. It is designed following an object-oriented approach that facilitates the creation of the sub models, their integration, and the application deployment on multiple platforms. The architecture of the application (see figure 15) is an analog to the tree structure that is used to define the RBAC policy.

The application supports the simultaneous activation of more than one role at a time and allows multiple selections of databases, tables, and tablefields. Allowing multiple fields selection creates the problem of congruency of the permitted operations on the selected field, because the user can have read access to all the fields but write access to only some of them. Therefore there will be the need to inform the users that certain actions cannot be executed because they are not allowed by the set of permissions for the specific role.

At the end of the user process, the system builds the query for the user and executes it. If the result is to be displayed, the system will display only the information that the user is cleared for, but if the operation is to write to the database or databases there will be the need to have different forms for different databases generated by the system to enable the modification.

Finally, the last step in the user process is either to display the results or to commit the changes to the databases. This step is done only after a final check by the temporal validation mechanism that indicates the validity of the user action, at that time, within the active set of roles and permissions. If the temporal validation mechanism detects a change in the authorization state the action is aborted and the user has to login again to the application and get the most current roles and relative permissions.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. JDRBAC APPLICATION DEVELOPMENT

A. INTRODUCTION

This chapter describes the design and creation of a Java-based prototype application for an RBAC policy that is applied across multiple database management systems. The intent of the prototype is to provide a proof of concept for Java Database Role-Based Access Control (JDRBAC). The prototype provides a working model and a foundation for future development of a practical application of an independent RBAC policy across loosely coupled databases.

B. TASK DEFINITION

The initial step in the development of the application was to define the distinct tasks that are to be performed, and determine the integration points for the separate tasks. It was determined that there existed five primary tasks that the application was to perform. These tasks are as follows:

1. Login – Each user must be required to authenticate themselves to the application via a user identification string and a password. This serves to as a means to restrict access to authorized users, and as a way to identify each user to allow for correct association of that user's assigned roles.

2. Interface – Once the user has logged into the application, an interface will be provided, displaying the appropriate data objects (database, tables, etc.) for the role or roles that the user has been assigned. The interface provides functionality for the user to select what data they wish to view, and submit their selection for retrieval from specific databases.

3. JDRBAC Tree Structure – When the application is launched, it will be necessary for the JDRBAC tree to be created in memory. This linked list structure will provide a ready representation of the RBAC policy that the application will enforce, without having to repeatedly access static memory. It will provide for search and retrieval of the composites of each role, so the interface can easily and quickly display the appropriate selections for each user.

4. Database Queries – Once the user has selected the data for retrieval, the necessary queries must be formulated and formatted, and performed on the proper databases to obtain the required data. The data must then be displayed for the user.
5. Reference – There must be a means of static storage of the information that is required by the application. This information includes the identification and passwords of all authorized users, and the role parameters of the RBAC policy. There must be a means to allow for other tasks (Login and JDRBAC Tree Structure) to access this data as required for their operation.

The generic operation of the application can be described as follows: The application is started, and the JDRBAC Tree is constructed from the reference data. A login prompt is displayed, and the user enters the appropriate identification and password. This information is verified with the parameters stored in the reference data. Upon user validation, the user interface is displayed, based on the current set of roles associated with that user. The role information is obtained from the JDRBAC tree structure. Once the user has made their selections on the interface, the necessary queries are performed on the appropriate databases, and the requested data is displayed for the user. The query process is repeated as required by the user.

The tasks and their interconnectivity are shown in the following diagram (figure 16):

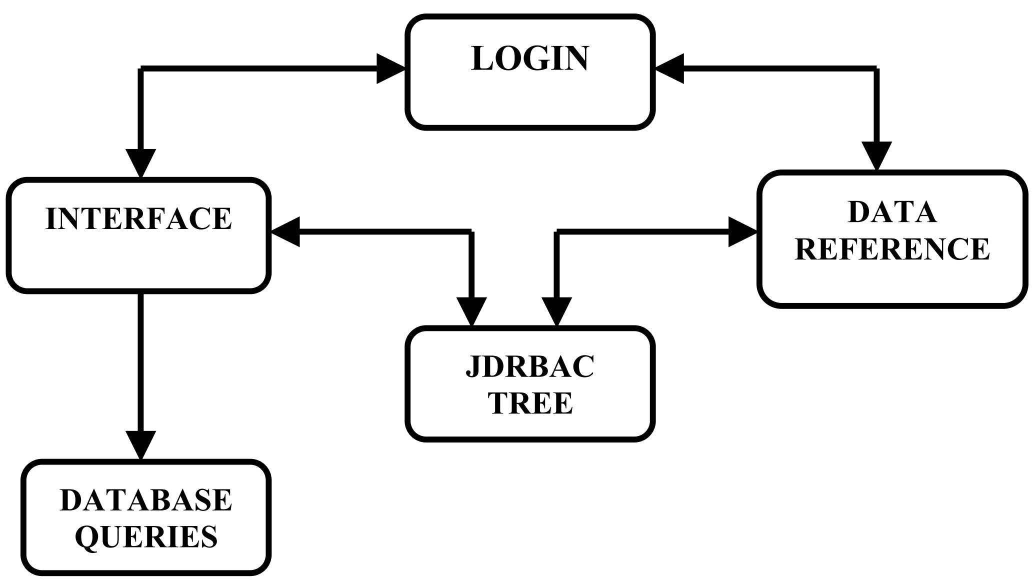

Figure 16. JDRBAC application tasks

One additional task was identified that was distinct and separate from the application itself. There must be a means for an administrator to act on the reference data. The necessary operations include the adding, deletion, or modification of roles, the association of users to roles, and the adding or removal of user login parameters. Because the actual administration of the reference data depends on the choice of the mechanism for storing that data, the manager task is directly related to the implementation of the application. A more detailed description of the manager interface will be included in chapter five, in conjunction with the description of the reference data storage.

C. CODE DEVELOPMENT

In keeping with an object-oriented design, each identified task was implemented in a separate Java class. This approach allowed for ease of development and testing of the code, as well as enabling the interface and the display of the query results to be tailored as desired without affecting the other modules.

To ensure low coupling with the application, each module was coded and tested independently. Once the proper operation was verified, the methods to interact with the other modules were included.

The remainder of this section consists of a description of the development of each module of the application:

1. Login

The Login module was coded to provide a pop-up window (figure 17) that provides the user fields to enter the user identification and password.

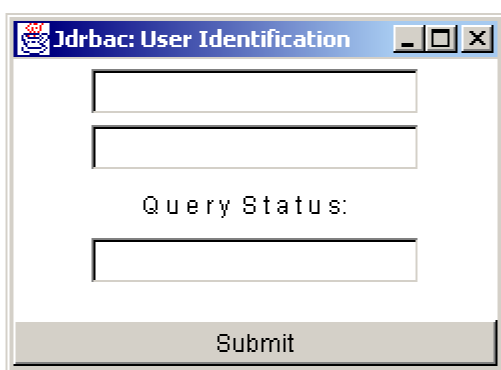

Figure 17. Login window

The window provides a status field to indicate incorrect entries and failed login attempts. An arbitrary number of login attempts, three, are allowed before the window closes and the application exits.

The login module passes the user ID and password strings to the reference module for validation. Once validated, it holds the user ID for access by the Interface module, and the login window is no longer displayed.

2. Interface

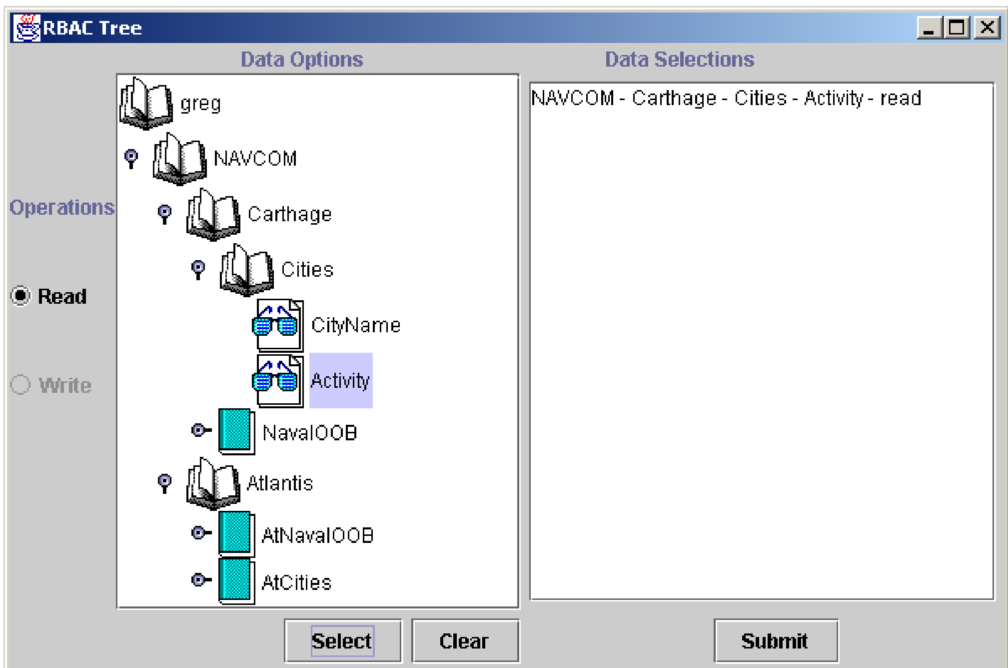

Figure 18. Interface window

The interface window (figure 18) was designed, for the purposes of the prototype, to display two main areas. On the left is a display showing the user name, which can be expanded to show the role or roles of the user. The user name is obtained from the login module, and the roles of that user from the Reference module. The left display can further be expanded to display to accessible databases of the role, the tables available in the specific database, and the tablefields of that table. The particular nodes that are displayed are obtained from the JDRBAC Tree Structure module, where the

representation of the RBAC policy is held in memory. The allowable operations (read or write) on each tablefield appear at the far left. The operations are only enabled when the user has selected a tablefield, and are not enabled when any other item is selected. At the bottom left of the window is a select button which, when a tablefield is highlighted, records the selection on the second main area on the right. There is also a clear button to remove the selection from the right display. Below the right display is a submit button. When the user has made a selection, clicking that button sends the parameters to the Database Queries module for data retrieval.

The selection of a tree structure on the interface was intended to mimic the linked-list JDRBAC tree in memory, to provide a demonstration of the JDRBAC tree. In an actual implementation the application could have any type of interface, including choices displayed with check boxes, or multiple screens.

The interface module was designed to contain the main method for the application. When the application is launched, the interface will call the appropriate methods to create an instance of the JDRBAC tree in memory, and instances of the other modules for the operation of the program.

3. JDRBAC Tree Structure

The JDRBAC tree is created in memory to provide a readily accessible means to access the RBAC policy representation. This was designed as a linked list structure that is created recursively through calls to the Reference module. The root of the tree is an arbitrary "JDRBAC" node. The first set of children are the defined roles, whose children are the databases associated with that role, whose children are the specific tables of that database, etc. The entire RBAC policy is represented in the JDRBAC tree. This has the advantage of preventing repeated calls to static memory via the Reference module.

Methods are included in the Tree module to enable searching of the tree for certain children. For example, given a role, the databases of that role are returned, given a role and a database of that role, the available tables of the database for that role are returned.

The creation of the JDRBAC tree allows for the entire RBAC policy to reside in memory, rather than requiring repeated calls to static memory.

4. Database Queries

This module was created to perform the queries to obtain the data requested by the user. It creates the Java Database Connectivity (JDBC) connections to the necessary database, and then executes the required query on that database. After completion of the processing of the returned data, the module creates a window for the display of the results (See figure 19).

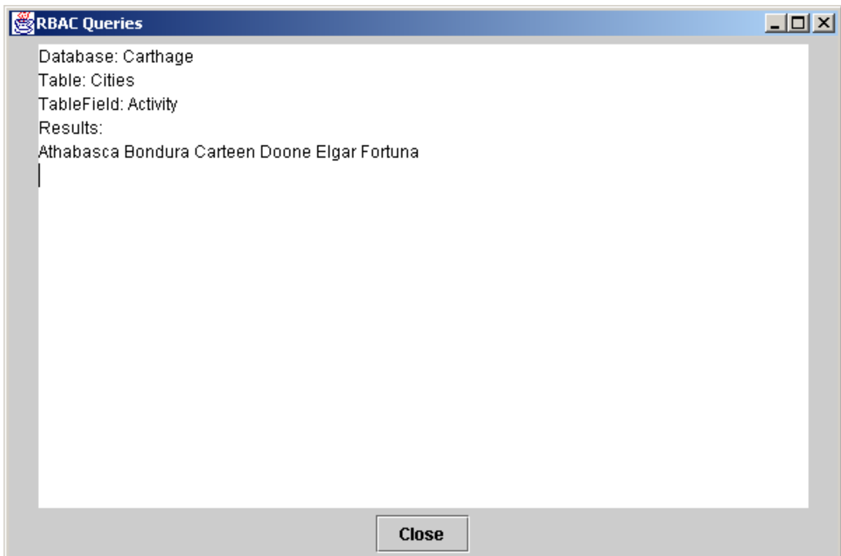

Figure 19. Result display window

5. Reference

The Reference module maintains the application data in long-term memory. The primary function is to provide a means for storage of the RBAC policy when the application is shut down. Although any number of means to store the data would have been sufficient, it was decided to utilize a database for storage of the data in order to simplify the administration of the reference information. In keeping with the Java implementation of the application, a pure Java database that could be directly integrated with the code was determined to be the best choice. After evaluating a number of open source applications, the small footprint, pure Java database created by the Hypersonic HSQLDB database open source project (the Hypersonic SQL Group coordinated through

the SourceForge.net web site) was selected. Further discussion of the Java database that was utilized is in the following chapter on implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. JDRBAC APPLICATION IMPLEMENTATION

A. INTRODUCTION

This chapter discusses the procedures for specializing (or tailoring) the JDRBAC application to address a particular problem. The primary step is the decision process for developing the RBAC policy that the application will enforce. A clear understanding of the access controls required is essential for an effective use of the JDRBAC application.

The chapter will also provide a detailed discussion of the Hypersonic HSQLDB pure Java database that was selected for the reference data storage, as well as the additional functionality and implementation options that the Hypersonic database provides.

B. UDERSTANDING THE RBAC POLICY OF THE APPLICATION

To fully take advantage of an automated implementation of the role-based access control policy that the application enforces, the application administrators need to understand the role-based access control policy models and rules. In particular, they should be able to view (or inspect) application structural charts as a set of relationships that tie each user to the task he is performing and each task description to the set of authorizations and permissions that the user can perform to accomplish his duties.

The advantage of managing access-control policy via JDRBAC is that the level of granularity and precision of the permissions can be refined to the full extent to regulate all types of access to the application data. Our prototype allows for controlling the access to the data based on the need to know of every specific user assuming a specific role. Additionally, the administrators need to select a role-based access control model that is appropriate for the application. For example, if there is need to completely separate the tasks performed by two different users in two different roles either statically or dynamically; the administrator should select the model that enforces the static separation of duties (SSD) or the dynamic separation of duties (DSD). Moreover, the application administrators need to define the role hierarchies and the role cardinalities based on the RBAC policy or on the actual way the tasks are being performed. This would allow for effective authorizations management and at the same time enforce an active security

mechanism that prevents abuse or misuse of the application resources. A clear, a priori understanding of the role-based access control policy model to implement, the rules and terminology, and the definition of a high level description of the application policy are prerequisites for being able to use the JDRBAC in an effective manner.

C. MAPPING THE APPLICATION POLICY

The high-level description of the JDRBAC application policy is a set of rules and constraints that provide a definition of the relationships between roles, resources and permissions. The next step is to map this high-level description into a data structure that is able to store all the possible authorizations. This step is composed of the following tasks:

- **Identify all users:** The administrator needs to have a list of all users of the application. The users do not have to belong to the same organization.
- **Assign identification and authentication credentials:** The administrator needs to assign a username that uniquely identifies the user to the application and an initial password that authenticates that user. The user, on the first login, can change the password, but the username remains fixed. The administrator has to follow the rules and guidance of a good identification and authentication policy in the assignment of the identification credentials.
- **Assign users to roles:** The administrator has to assign the users to roles to reflect the actual application policy. During the user-role assignment the administrator has to consider the role hierarchies and the cardinality of each role to detect and correct errors.
- **Define the role-database relationships:** The definition of a relationship between a role and a database requires that the users assuming the role are allowed access to the database in the relationship. Each relationship is an authorization that certifies that the role is globally authorized to access the information in the database. The absence of such a relationship means that no access can be provided to the user. At this step compartmentization and need-to-know policy can be enforced.

- **Determine the role-database-tables relationships:** At this point the administrator needs to define the tables of every database that a specific role is allowed to access. Even from this step we can see how the JDRBAC policy implementation allows for more granular authorization enforcement. This differs from database management systems that are based on discretionary access-control policy enforcement, and that allow access to all or nothing. The absence of a certain relationship means that the role is not allowed to access the data that is stored in the tables of the database. Moreover, the administrator has a tool to define the access to multiple databases in a distributive context.
- **Determine the role-table-tablefields relationships:** The next level of granularity of authorizations that the JDRBAC policy implementation enforces is the definition of the tablefields that the role is authorized to access within the parent table. The size of the set of relationships can grow dramatically if the tables have many tablefields, but a modular design of the database and a systematic approach would alleviate the authorizations management burden.
- **Define the role-tablefields-operations:** Finally, the last level of granularity consists of the administrator defining the operations that a specific user can exercise on a specific tablefield of the database. The operations that are valid in the application are the "read," in which the user can have access to the value of the data in the tablefield; the "write," in which the user can change the value of the data in the tablefield without having the authorization to see it, or the "read & write," in which the user reads the value in the tablefield and changes it.

D. STORAGE OF THE JDRBAC REFERENCE DATA

The sets of the previously described relationships form the core of the role-based access control policy implementation. Within the JDRBAC application, the RBAC policy is actualized in memory within the JDRBAC tree structure. When the application itself is not running, there is a requirement for an optimal way for the administrator to store the

application JDRBAC policy. One solution is to create a database, although storage could be done by text files or other types of storage. Choosing a database for the reference data storage provides for an easier means to administer the stored information.

Given the option of creating a database for the reference data, the administrator needs to select the correct database management system in which to implement the authorizations database. In light of the Java-based application, the best choice for integration with the application would be to consider a pure Java database that would hold the small authorization database. There are a number of such databases available via open source. After some evaluation, it was decided to use the Java database developed by the Hypersonic HSQLDB open source project found at SourceForge.net. The following is a brief description of the Hypersonic database.

1. Hypersonic SQL/ HSQLDB:

The Hypersonic SQL/HSQLDB is an open source Java database engine. It is a general-purpose relational database that can be used as an applet or as a distributed application. It is a very small and easy to install database originally intended for use on handheld devices. The Hypersonic SQL/HSQLDB has a standard SQL syntax that an administrator is required to know in order to manage the database. Additionally, it has a Java-Database Connectivity (JDBC) interface that supports the creation of connections to other vendor's databases.

The database is simple and compact, which makes it suitable for the JDRBAC policy storage. Moreover, it can be used either in standalone mode or in client-server architecture.

The Hypersonic SQL/HSQLDB database has a built-in security system that manages user names, passwords, and access rights. There exists by default a 'System Administrator' with the user name 'sa' and an empty password. This special user can create new users, drop users, and grant and revoke access rights for tables to other users.

E. DESIGN OF THE RBAC POLICY REFERENCE DATA

Having selected a database management system that will store the RBAC policy the application administrator has to design the authorization database. In particular, this step consists of defining the metadata (or the schema) of the database. Next, the

administrator has to define the primary keys, the foreign keys, and the index tables in a way that supports requirements for referential integrity and guarantees the best performance and normalization of the database.

The administrator can use the standard SQL syntax or scripts to create the authorization database. A screen snapshot of the java interface, as developed by the Hypersonic SQL Group, of the Hypersonic SQL database manager is shown in figure 20.

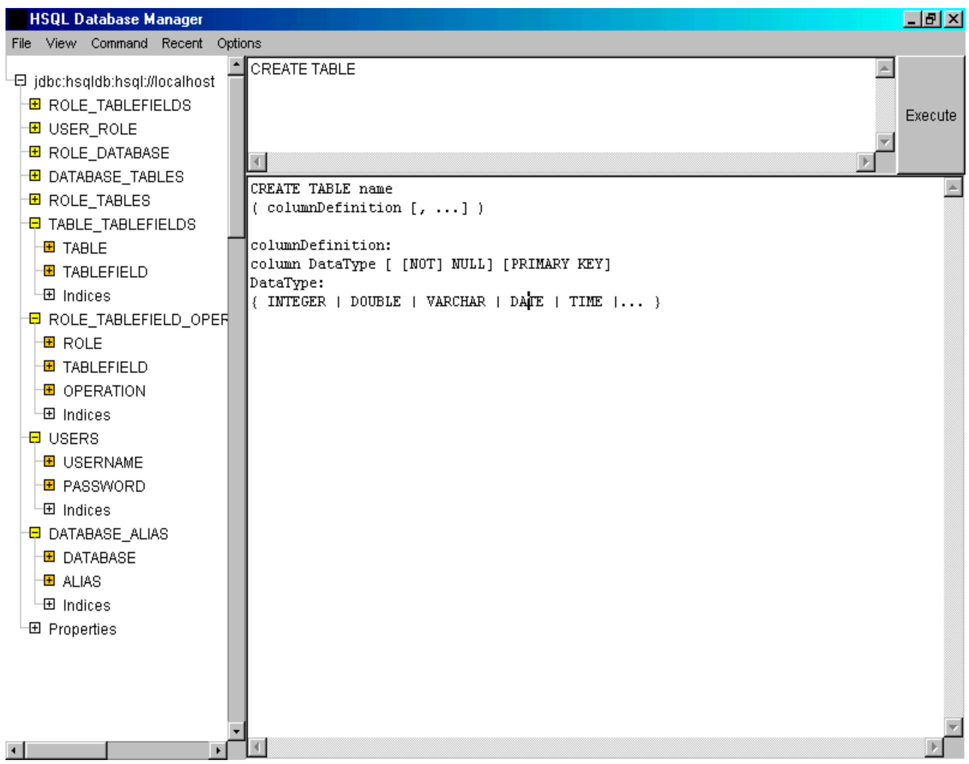

Figure 20. Administrator interface

When using the Hypersonic database in the client-server mode, the first step is to launch the server by executing a batch file named “runServer.bat”. Next the administrator needs to execute a second program, which is “runManager.bat”, and provide the correct credentials and JDBC driver to connect to the database and run the database manager. In the case shown, the server and the client are on the same machine; therefore we connected using the “://localhost” URL.

After creating the database schema, the administrator has to input all the data in the tables to reflect the RBAC policy of the JDRBAC application. Finally, the administrator can prepare Java methods that query the Java database and return results to other calling programs. The list of these methods is given in table 1:

Method	Description
validateUser(username, password)	Returns true if the user is identified and authenticated
getRoles(username)	Returns a vector of roles that the user can assume
getDatabase(role)	Returns a vector of all possible databases that the user is allowed to access
getTables(role,database)	Returns a vector of the tables of the database that the role is allowed to access
getTableFields(role, database, table)	Returns a vector containing the allowed tablefields
getTablefieldOperation(role, tablefield)	Returns a vector of the operations that the user assuming that role is allowed to perform on the tablefield.

TABLE 1. Query methods

F. THE ADMINISTRATOR INTERFACE

The drawback in building the reference data using the hypersonic database is the requirement for the administrator to have an understanding of SQL. Moreover, the management of the reference data requires extensive operations to accomplish even the simplest tasks. To alleviate the authorizations management burden and to improve the correctness of the reference data, an additional interface has been created. This interface allows the application administrator to manage the sets of users, roles, resources, and grant or revoke authorizations. Figure 21 is a screen snapshot of the administrator interface that shows the operation of assigning a user to a role.

Such operation would otherwise need the supply of an SQL statement in the Hypersonic administrator interface.

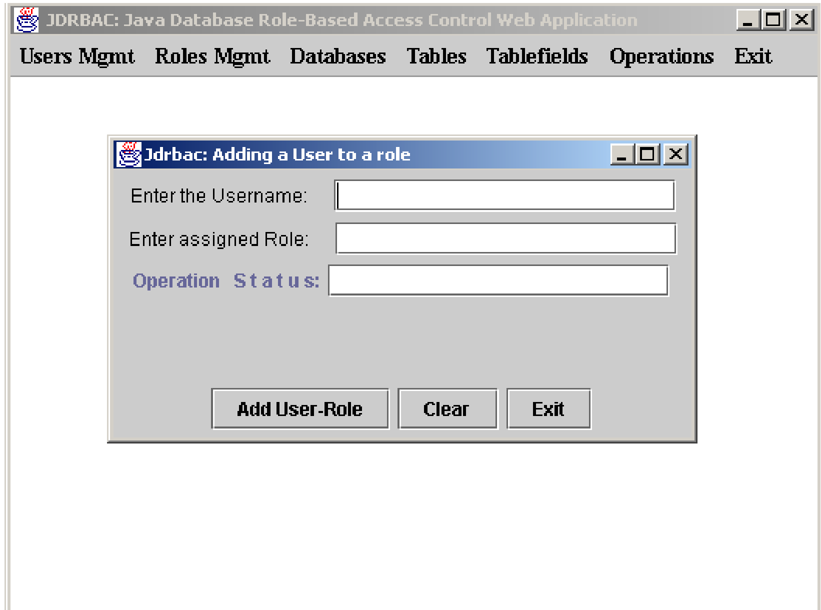

Figure 21. Application administrator interface

The administrative interface was developed for interaction with the Hypersonic database. But the code was written to allow for connectivity to other databases, should another one be used for the reference data storage. By modifying the code to reflect different Java connectivity parameters, the interface can be used to provide a simple, intuitive management mechanism for any other database. This assumes that the RBAC policy development follows the previously outlined procedures.

G. TESTING

There are many ways that the database administrator can ensure the correctness of the database. For example, the administrator can use SQL queries to check the validity of the results. Another way is to write a Java program that tests the validity of the RBAC policy implementation. The testing phase is needed in order to detect errors in the database or in the actual RBAC policy design and to prevent damage from erroneous

policy that could result in untended flows of data (i.e., those flows that are contrary to the policy that is supposed to be enforced).

H. ADDITIONAL FUNCTIONALITY

The RBAC policy of the JDRBAC application is represented by the linked-list tree structure within the application. The tree holds the entire RBAC policy, which alleviates the need to consistently make calls to static memory during the operation of the JDRBAC application.

The choice of a pure Java database for storage of the RBAC policy reference data allows for the database to be stored in memory during application operation. In this sense, the database provides redundancy to the JDRBAC tree. This provides additional flexibility in implementing the application. The JDRBAC tree structure would not be required to store the entire RBAC policy, but could be tailored to store only the portion of the RBAC policy that applies to the current user. When the user logs in to the application, an instance of the JDRBAC tree can be created for that user based on the reference data stored in the Java database. This would reduce the level of computer resources required for operation of the application, and provides a means for improving performance of the application.

VI. SECURITY

A. INTRODUCTION

Security of the data flows is a primary concern when fielding the JDRBAC application in a real-world situation. It makes little sense to attempt to enforce an access control policy if the data is passed in the clear, where unauthorized sources can intercept and collect the data. Therefore, some form of security mechanism is necessary to provide appropriate levels of confidentiality and integrity for the data being processed by the JDRBAC application.

B. DISCUSSION

In a real-world situation, the data channels between the users and the application, and between the application and the databases, will likely be across the Internet (figure 22). As these channels are exposed to unauthorized monitoring, it will be necessary to incorporate encryption of the data to provide minimal levels of security. As part of the application development, a number of possible methods were examined to integrate the encryption mechanisms into the JDRBAC application.

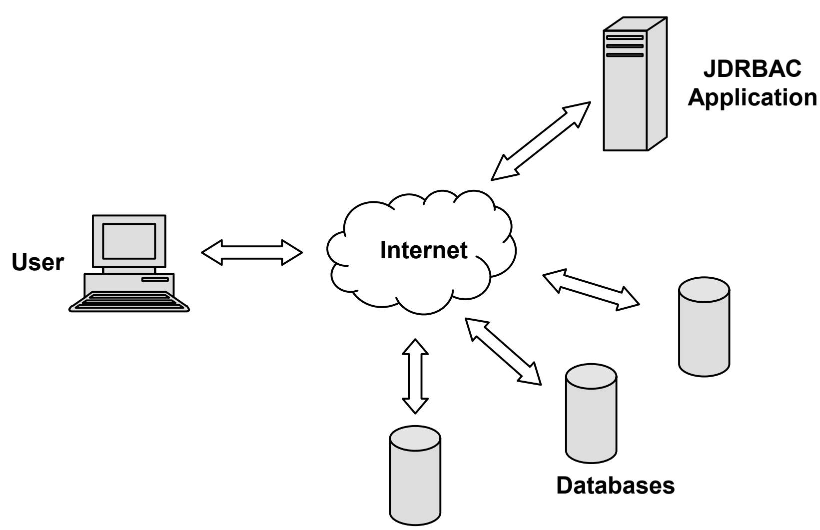

Figure 22. Application deployment

C. CLIENT – APPLICATION SECURITY

For the data flows between the user and the application, the use of Secure Sockets Layer (SSL) is the best choice (figure 23). This is due to the existing support for SSL that is included in most web browsers. By using SSL, there is no need to additional software or functionality to be provided to the users. SSL provides authentication and data integrity, and can be used to secure communication between a web server hosting the JDRBAC application and a web browser on the user's computer. Sun's Java Secure

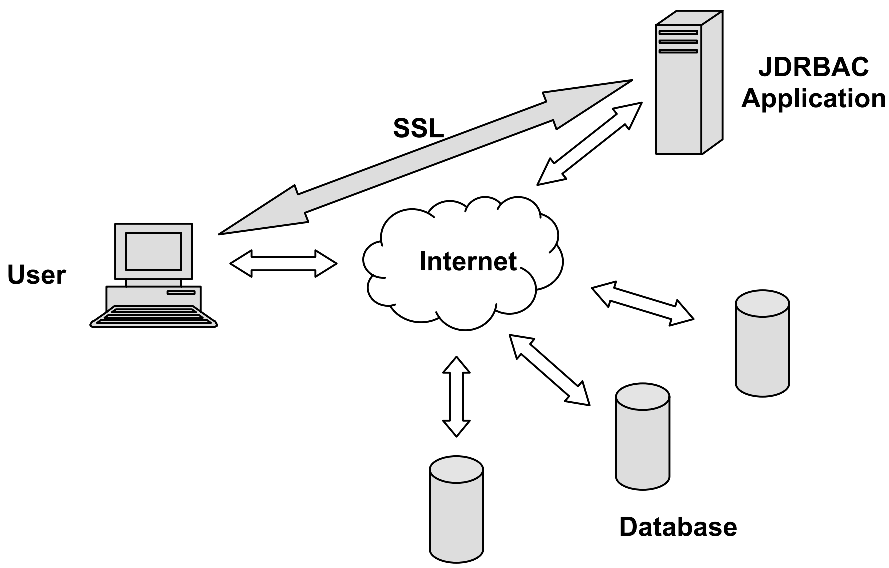

Figure 23. Securing the application with SSL

Sockets Extension (JSSE) provides a freely distributed SSL implementation for Java applications.

When SSL is used for encrypted connections via web servers, the client authenticates the web server, while the web server allows any client to connect. In the JDRBAC application, the user is authenticated by the user identification/password combination in order to provide client validation. Should further client authorization be desired, the JSSE can provide authentication via a valid certificate possessed by the user, if such a public key infrastructure is available.

D. DATABASE – APPLICATION SECURITY

The integration of encryption between the JDRBAC application and the various databases that are accessed requires a more complex solution. Any encryption mechanism introduced will require a means to encrypt and decrypt data at the application and at the database.

One method involves the use of the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE). The JCA and JCE were designed by Sun to provide an implementation-independent Application Programming Interface (API) for cryptographic functions in Java. The JCA provides message digests and digital signatures, and the JCE adds support for ciphers such as DES, TripleDES, and Diffie-Hellman.

The inclusion of the JCA and the JCE allow for the use of password-based encryption between the application and the database (figure 24). In this method, the application performs encryption of the database query using a shared static key, and sends it to the database. A separate application then decrypts the information with the same shared key. The separate application would then perform the query, encrypt the results with the shared key, and return them to the original application.

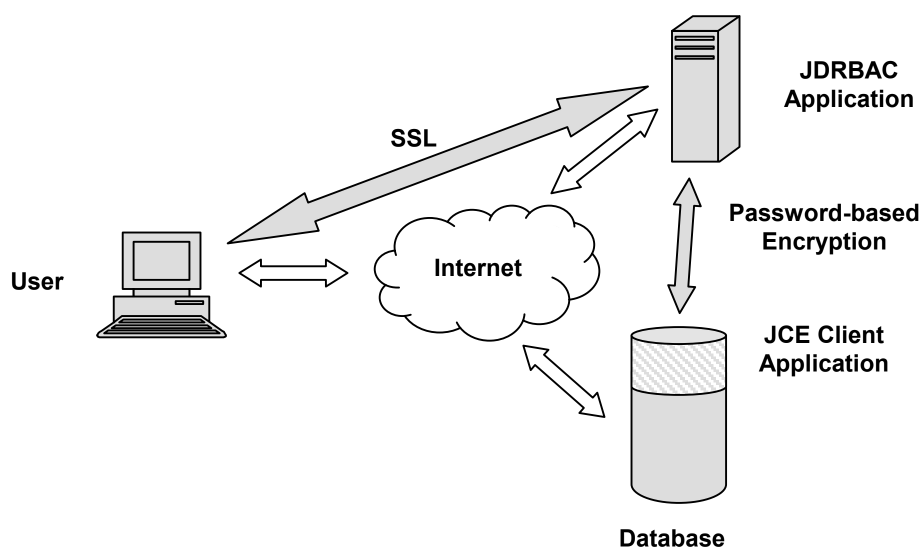

Figure 24. Adding encryption

There are issues that arise when using password-based encryption. First, there is the need to have a separate password for each application-database connection, in order to ensure that compromise of one password and the associated data channel does not induce the same compromise of the other channels. The password stored for a particular database would be at some risk of disclosure to unauthorized users. In addition, all of the passwords would have to be stored on the server hosting the JDRBAC application, inducing overhead within the application, and providing a central point of risk for the compromise of all passwords.

Also, there is the requirement for a client application at the database to perform the encryption and decryption. The administrators of the remote network where the database resides may not be willing to accept the inclusion of such an application within their networks, especially if it resides on the database server.

A second, potentially more desirable method, involves using the JSSE to perform Remote Method Invocation (RMI) using SSL. In this case, a secure Java Database Connectivity (JDBC) driver accepts the JDBC requests from the application, and communicates with a secure JDBC server on the database using encryption to protect the data. An illustration is the process is in figure 25.

While this SSL implementation can be performed directly from the server that is hosting the JDRBAC application, for purposes of scalability and performance improvement, a proxy server can act as middleware to perform the database connections (figure 26). Whether or not to include a middleware element would depend on the number of users that will access the JDRBAC application (particularly the number accessing the application concurrently), and the number of databases to which the JDRBAC application will interface.

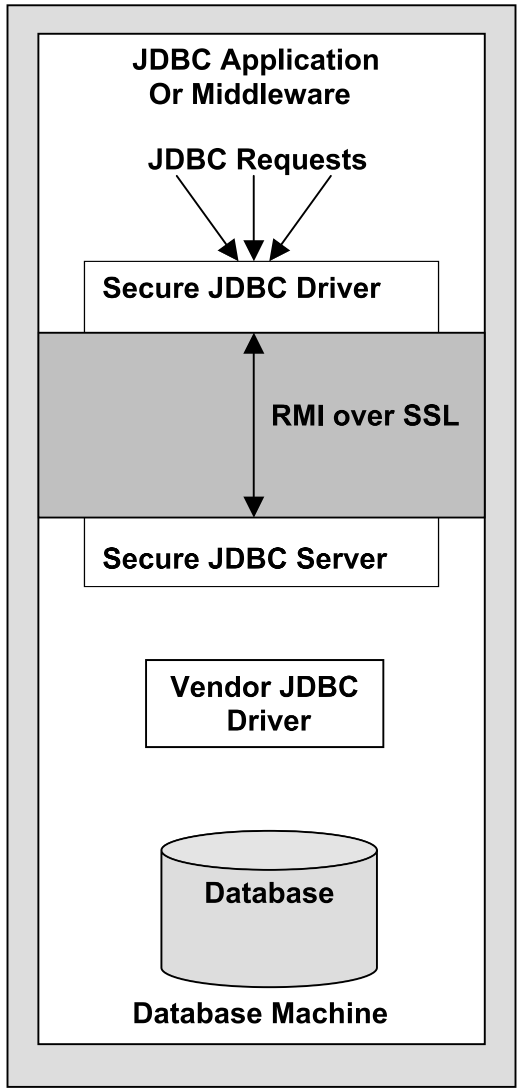

Figure 25. Using JSSE to secure the application

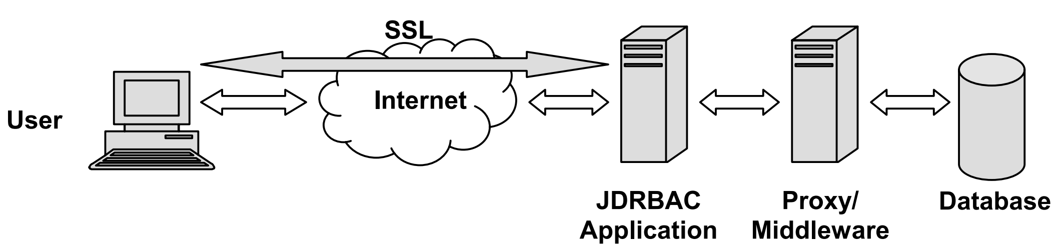

Figure 26. Use of a proxy server

The method of RMI using SSL has the advantage of not requiring a static password to be stored, since SSL uses a session key instead of a password. It also

provides potential support for the use of certificates if a public key infrastructure is in place.

Like password-based encryption, there is still the requirement for a separate application to reside on the database side (in this case a secure JDBC server). But this is unavoidable if encryption is to be performed on the data channel between the application and the database.

A third solution would be to use a Virtual Private Network (VPN) solution to secure the channel, but the expense of implementing such a solution makes it the least desirable one.

E. DATABASE SECURITY

In addition to securing the channels of communication, there is the need to secure the database itself. But since this is not part of the functionality of the JDRBAC application, it will be left to the database administrators and their associated network administrators to provide for sufficient security.

F. APPLICATION SECURITY

Finally, there is the need to provide for adequate security of the JDRBAC application itself. Because the application will be processing data internally in the clear, it is necessary to ensure that an unauthorized user does not gain access to the server where the application resides and be able to collect and view data. It is therefore important that the web server where the application resides be configured securely. Once again, the configuration of the web server is beyond the scope of the JDRBAC application functionality, and so will be left to the administrators responsible for the web server (though they will in all likelihood be the same people responsible for the installation of the JDRBAC application).

VII. CASE STUDIES

A. INTRODUCTION

This chapter discusses three cases in which an application-level database RBAC (DRBAC) solution can be applied, and for which the implementation of such a solution may be more desirable or effective than the alternatives. The importance of these cases is that they provide a validation of the concept presented in the thesis in real-world situations, and they help to provide a focus for future research on RBAC.

B. DISCUSSION

The first case study is drawn from an article in the Wall Street Journal published on November 13, 2001. This article describes the desire of health agencies and medical personnel to view the inventory and sales tracking data of various pharmaceutical companies. The health agencies' main goal is to be able to identify spikes in the sales of certain drugs, in that such higher-than-normal demand can be an indicator of potential epidemics.

In this situation, the users (health agency and other medical personnel) have a desire to view only certain subsets of the various databases. They do not need to search the database, or view data beyond that which is associated with a certain drug or drugs produced by certain pharmaceutical companies. So there are static subsets of information that is of interest.

Due to privacy and security concerns, the direct integration of the databases, or the consolidation of the data into a single DBMS, are not valid options. Competing companies are not going to integrate their sales tracking or inventory data with each other within a single DBMS. For the same reasons, a SAP or Tivoli type of solution is not a likely option. Providing the users with individual accounts on the databases of interest would create a great deal of overhead for the companies involved, and expose the companies' data to increasing levels of risk of unauthorized disclosures.

An application such as DRBAC provides an alternative to the foregoing types of solutions. The companies would provide a single account for the application, rather than accounts for each user. The account would be restricted to specific information,

providing a sense of assurance that users will not be able to exceed the desired boundaries established. Moreover, the users are provided with a single point of access to the desired information, rather than needing to access each company's data separately.

The RBAC policy enforced by the application would not just restrict the access of users, but provide a tangible benefit. Roles could be created for distinct drugs, and users would be assigned roles based on the drugs they wished to track. This would allow for the users to only view their data of interest, rather than having to search through all the information for every drug represented through the application account access.

A second case is a joint military operation involving a particular area of interest. Such an operation would be of a limited time in duration, and can involve multiple services, or multiple companies. The operation would be focused on a particular area of interest. Examples of such operations would be joint exercises, Non-combatant Evacuation Operations (NEOs), or humanitarian relief efforts. The operations may be well-planned, or require support establishment in a short period of time.

The data of the interest on the particular area (e.g. meteorological data, order of battle data, and intelligence data) could be managed by distinct DBMSs. For an operation of limited duration, the costs of creating a centralized database or implementing a SAP-type of solution may be prohibitive. In addition, there may not be adequate time for the creation of such solutions. In the case of a joint operation, directly integrating databases from separate services (or from multiple countries) would not be economical or not possible due to security concerns. In addition, time may still be an issue.

A database RBAC application provides the flexibility and ease of adaptation that can make it useful in such environments. It provides access to multiple users across multiple databases with relatively low overhead and cost. The need for a static subset of data for the application to access precludes the use of dynamic searches of the databases, which may restrict the application's usefulness for tactical purposes. But for support purposes, the application can provide for the required functionality.

The RBAC policy implementation of the application provides for access to be partitioned based on need-to-know (e.g. by nationality: U.S, second party, or third party allies) or by security clearance. Roles can be created based on the security level of the

data to be accessed. The access control can also be based on the individual, unit or command functions, with roles devised based on the tasks or missions that are to be performed.

The single point of access to the various databases (via the application account) allows for necessary security controls and restrictions to be put in place on each DBMS to which the application is granted access. The user validation mechanism of the application ensures that users will be restricted to specific accesses as negotiated by the responsible parties (the DBMS owners, the application administrators, and other entities with controlling interest or responsibilities). It also provides better user support in a data-rich environment through consolidation of multiple sources of information. As user-specified requirements change, the ability to add additional sources of information (in the form of additional DBMSs accessed by the application) is also an asset.

The third case involves the current anti-terrorism environment that exists in the United States. There is a need for security personnel (e.g. border patrol, immigration agents, and airport security) to identify individuals with possible terrorist motives. The information on such personnel is located on various databases (e.g. maintained by the FBI, CIA, local law enforcement, INTERPOL). There is a need to enable personnel at various points of entry into the United States with the ability to identify terrorist suspects prior to their entry into the country.

One solution would involve the creation of a central database that will hold the necessary information, with updates provided by the various law enforcement agencies. But the expense and labor required for designing and maintaining such a centralized DBMS is not necessary. The DRBAC application can provide the consolidation of the access to the data within the various distinct law enforcement DBMS.

The agencies (FBI, CIA, etc.) can create key lists of suspects that should be stopped, detained, or identified. The DRBAC application can be granted access to the database tables that contain these keys lists. The tables themselves can be regularly updated by the various agencies without the need to alter the application access that has been given.

Personnel at the entry points can access the DRBAC application via the Internet, and check the identification of incoming travelers against the various lists. Roles can be created based on the tasks of the various security personnel utilizing the DRBAC application. Personnel first checking the identification can be provided the minimal data (e.g. name, description, and photograph) to flag suspects. The suspects can then be turned over to other security personnel for further investigation. These other personnel can be given roles that allow for more detailed information from the various databases (e.g. aliases, prior destinations, background) that will enable them to positively identify and detain terrorist suspects. Additional roles can be created for police and highway patrol throughout the country, to allow for them to check arrestees and traffic stops against the key-suspect lists.

These scenarios provide only a few of the ways in which a DRBAC application can be applied. The flexibility, low-cost, and ease of implementation make it a valuable tool for situations in which the data is held on various distinct and distributed databases, the subsets of data are well-defined, and the access to the data should be mediated by an access control policy.

VIII. CONCLUSIONS AND FUTURE WORK

Role-Based Access Control has gained acceptance in several information technology areas. Although RBAC mechanisms have been incorporated in many database management and operating systems, there is strong need for RBAC policy implementations that are vendor neutral and platform independent, especially in a massive distributed environment. In this thesis, the U.S. National Institute of Standards and Technology (NIST) standard reference model was examined to present a foundation for an RBAC policy implementation. In particular, the work of this thesis constitutes an attempt to implement a Java Database Role-Based Access Control (JDRBAC) application in a distributed environment. The distributed environment can be defined as one in which databases span various enterprise boundaries and associated trust domains. The JDRBAC application proposed in this thesis constitutes a model for enforcing role-based access control on multiple loosely-coupled databases that can belong to different enterprises and can be as heterogeneous as possible. The advantages of such an implementation are the streamlining of the authorization-management process, the access transparency provided to the users, the flexibility and adaptability of the application to many scenarios in which high degrees of collaboration and rapid deployment are required, and the scalability and interoperability offered by the application are important.

The JDRBAC application implements a general-purpose mechanism that is platform independent and that will allow the negotiation of the user access to distributed databases without the need to have pre-assigned accounts on any of the database servers. It provides a higher level of control that allows for a higher degree of flexibility in the administration of privileges. The application is composed of four modules. First is an authentication and identification module that checks the user’s credentials and associates a user with their respective roles. The second module represents the RBAC policy implementation, modeled as a tree structure capable of storing the entire RBAC policy in memory. The JDRBAC tree is created from reference data that is maintained in a pure Java database. For the purpose of this thesis, an open source database (the Hypersonic Hsqldb Database Engine) was used to model the set of authorizations and permissions for every database access. The third module consists of the JDRBAC web application that

permits the users to login and then submit queries on the information that they are authorized and allowed by the activated roles. The last module is the temporal validation mechanism that accounts for the temporal aspects of the application such as the activation of multiple roles at the same time, the detection of change in the authorizations state relative to the current user and the choice of dynamic actions in case of revocation or policy change.

In the development of the prototype, the Java language was selected because it allows for platform independence, flexibility, and modular approach, but such a prototype could have been built using other languages such as ADA, or C++. The choice of the programming language is not critical to the demonstration of the usefulness of the JDRBAC implementation model.

The reference data allows the storage of the set of authorizations and permissions maintained in the JDRBAC tree structure. The implementation of the reference data uses an open source prototype entirely developed in Java by the Hypersonic HSQLDB database open source project coordinated through the SourceForge.net web site. The choice of this product is not vital for the JDRBAC application, but provides an excellent implementation based on a specific vendor DBMS. Ordinarily, to use the database, the administrator needs to have some experience with Java and SQL. To get around this requirement, an administrator interface was built and integrated with the JDRBAC application, to make administrative tasks much simpler and less cumbersome than without the interface.

The prototype built for this thesis is a working model that is intended as a reference implementation for database role-based access control. The model has not been perfected, leaving many aspects for improving upon the model future research.

First, the JDRBAC tree structure is designed to load the entire RBAC policy in memory. Future research is needed to study the feasibility and scalability aspects that result from this design choice. In particular, the prototype was tested using a small set of roles, and databases with a small number of tables and tablefields. Future research can determine the extent and limitations of this model in a vast distributed environment.

Second, the set of operations allowed for the users in the JDRBAC implementation are limited to read and write. The extension of the set of operations to allow users to create or delete objects from the databases raises issues of concurrency control, especially when two or more users are editing the same information. The study of the concurrency control mechanisms suitable for the JDRBAC implementation is left to future research.

Third, the JDRBAC implementation does not account for dynamic detection of policy changes, such as permission revocation. Neither does it address the simultaneous activation of multiple roles. These two aspects are crucial for the efficient operation of this application in a distributed environment. The study of the temporal validation mechanism is left to future research.

Finally, The JDRBAC application is intended to be used in an open environment such as the Internet. Therefore special attention has to be given to the security of the application and the communication channels utilized by the application. These channels include both those between the application and its user community, and between the application and the database systems it accesses. Brief solutions are presented in the security chapter that propose the use of secure socket layers (SSL) combined with encryption, but a more profound security study remains to be conducted.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Barkley, J., Beznosov, K., Uppal, J. (1999) Supporting Relationships in Access Control using Role-Based Access Control. In *Proceedings of the fourth ACM workshop on role-based access control*, October 1999.

Barkley, J., Kuhn, D.R., Rosenthal, L., Skall, M., Cincotta, A. Role-Based Access Control for the Web. National Institute of Standards and Technology Gaithersburg, Maryland 20899.

Bertino, E., Bonatti P.A., and Ferrari, E. (2000) TRBAC: A temporal role-based access control model. In *Proc. Fifth Workshop on Role-Based Access Control*, ACM (Berlin, Germany, July 2000), 21-30.

Beznosov, K. (2001) Engineering Access Control for Distributed Enterprise Applications. Center for Advanced Systems Engineering School of Computer Science, Florida International University.

Beznosov, K., Deng, Y. A Framework for Implementing Role-Based Access Control Using CORBA Security Service. Center for Advanced Systems Engineering School of Computer Science, Florida International University.

Beznosov, K., Deng, Y., Blakley, R., Burt, C., Barkley, J. (1999). A resource Access Decision Service for CORBA-Based Distributed Systems. In *Proceedings of the Annual Computer Security Applications Conference*. (Phoenix, Arizona, USA, December 6-10, 1999).

Bhasker, B., Egyhazy, C. J., Triantis, K. P. (1992) The architecture of a heterogeneous distributed database management system: The distributed access view integrated database (DAVID). In *Proc. Annual Computer Science Conf.*, ACM (Kansas City, Mo., April 1992), 173-179.

Eig, J., Burton, T.M., (2001, November 13). Drugstore data could be tip-off to bioterrorism. *Wall Street Journal*, p. B1.

Ekstein, R., Loy, M., Wood, D. (1998). *Java Swing*. Sebastopol, CA: O'Reilly and Associates.

Espinal, L., Beznosov, K., Deng, Y. (2001) Design and Implementation of Resource Access Decision Server. Center for Advanced Systems Engineering School of Computer Science, Florida International University

Ferraiolo, D., Barkley, J., Kuhn, D.R. A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet. NIST Gaithersburg, Maryland 20899.

Ferraiolo, D., Kuhn, R. (1992). Role-Based Access Control. In *Proceedings of the 15th National Computer Security Conference*, NIST Gaithersburg, Maryland 20899.

Flanagan, D. (1999). *Java in a Nutshell, Third Edition*. Sepastopol, CA: O'Reilly and Associates.

Garms, J., Somerfield, D. (2001). *Professional Java Security*. Birmingham, UK: Wrox Press Ltd.

Mönkeberg, A. and Rakete, R. Three for one: role-based access-control management in rapidly changing heterogeneous environments. In *Proc. Fifth Workshop on Role-Based Access Control*, ACM (Berlin, Germany, July 2000), 83-88.

Sandhu, R. Engineering Authority and Trust in Cyberspace: The OM-AM and RBAC Way. ISE department, George Mason University.

Sandhu, R., Ferraiolo, D., and Kuhn, D. R. The NIST model for role-based access control: Towards a unified standard. In *Proc. Fifth Workshop on Role-Based Access Control*, ACM (Berlin, Germany, July 2000), 47-63.

APPENDIX A

JDRBAC SOURCE CODE

This appendix contains the source code for the JDRBAC prototype created as a proof of concept for the thesis. Although the prototype is a fully functional application, it does not consist of a finished program for distribution. The code is presented in an "as-is" format, intended only as a foundation for future research. The code references, but does not contain, source code from the Hypersonic SQL Group.

THIS PAGE INTENTIONALLY LEFT BLANK

```
//--------------------------------------------------------
// Filename:    Login.java
// Date:        02/24/2002
// Project:     JDRBAC implementation
// Compiler:    SDK 1.3
//--------------------------------------------------------

package rbactreetest;

import java.sql.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.tree.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

/*
 *  This class is used to identify and authenticate
 *  the user of the JDRBAC application implementation.
 *  The user has to provide a username and a password as
 *  credentials. Only three attemps are allowed.
 *
 * @authors Greg Nygard; Hammoudi Faouzi
 *
 */

public class Login extends Frame implements ActionListener
{

  /* *****************************
    Data members
  *******************************/

  private TextField nameField;
  private TextField pwdField;
  private TextField status;
  private String temp;
  private String userName;
  private String password;
  private int counter=0;
  private Button enter;
  private Panel pN;
```

```
private boolean validLogin;

private ResultSet rset;    // JDBC connection parameter
private Statement stmt;    // JDBC connection parameter

// ----------- -----  Connection Variables ------------------

String userid = "sa";
String pass = "";
String url = "jdbc:hsqldb:hsql://localhost";
String driver = "org.hsqldb.jdbcDriver";
Connection conn;
// ----------- ------------------------------------------------

/********************************
   the Constructor
********************************/

public Login()
{
  super("Jdrbac: User Identification");
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
       System.exit(0);
    }
  });

  pN = new Panel();
  nameField = new TextField(20);
  pwdField = new TextField(20);
  nameField.addActionListener(this);
  pwdField.addActionListener(this);
  pwdField.setEchoChar('*');
  pN.add(nameField);
  pN.add(pwdField);
  enter = new Button("Submit");
  enter.addActionListener(this);
  add(pN);
  pN.add(new Label("Q u e r y  S t a t u s:"));
  pN.add(status = new TextField(20));
  add("South",enter);
  setBounds(350,200,250,180);
  setVisible(true);
```

```
  // Connecting to the database.
  try
  {
    Class.forName(driver);
    conn = DriverManager.getConnection(url, userid, pass);
    stmt = conn.createStatement();
  }
  catch(ClassNotFoundException cnfe)
  {
    System.err.println(cnfe);
  }
  catch(SQLException sqle)
  {
    System.err.println(sqle);  // error connection to the database
  }
}

/****************************************
   Public Methods:

    void        actionPerformed(ActionEvent evt)
    String      getUserName()
    boolean     validLogon()
****************************************/

/************************************
   this method checks the validity of
   the username and password provided
************************************/

public void actionPerformed(ActionEvent evt)
{
  Object source = evt.getSource();
  if(source.equals(enter))
  {
    try
    {
      stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                  ResultSet.CONCUR_READ_ONLY);
      rset = stmt.executeQuery("select * from users " +
        "where users.username = '" + nameField.getText() + "' " +
        "and users.password = '" + pwdField.getText() + "'");

      if (rset.next())
      {
         userName = (rset.getString(1));
```

```
          password = (rset.getString(2));

          status.setText("Valid login ");
          validLogin = true;
          counter = 0;
        }
        else
        {
          counter ++;
          status.setText("Incorrect Login  "+counter);
          validLogin = false;
          if(counter==3)
          {
            dispose();
            System.exit(0);
          }
        }
      }
      catch(SQLException sqle)
      {
        status.setText(sqle.getMessage());
      }
    }
  }

  /*******************************
     this method returns the username
     of the user logging in
  *******************************/

  public String getUserName()
  {
    return userName;
  }

  /*******************************
     this method returns true
     if the login is valid
  *******************************/

  public boolean validLogon()
  {
    return validLogin;
  }
}
```

```
//------------------------------------------------------
// Filename:    Drbac.java
// Date:        02/24/2002
// Project:     JDRBAC implementation
// Compiler:    SDK 1.3
//------------------------------------------------------
package rbactreetest;

import java.sql.*;
import java.util.*;

/*
 *  This class is used to provide the JDRBAC tree with
 *  the reference data. It connects to the Hypersonic database
 *  and retrieves all the needed information on the particular user.
 *
 * @authors Greg Nygard; Hammoudi Faouzi
 *
 */

public class Drbac
{

  /*******************************
   Data members
  *******************************/

  public static String username;
  public static String password;
  public static boolean validUser=true;
  public static String sRole;
  public static String sDatabase;
  public static String sTable;
  public static String sTableField;
  private ResultSet rset;
  private Statement stmt;

  // ----------- -----  Connection Variables ------------------

  String userid = "sa";
  String pass = "";
  String url = "jdbc:hsqldb:hsql://localhost";
  String driver = "org.hsqldb.jdbcDriver";
  Connection conn;
 // ----------- ------------------------------------------------
```

```
/*******************************
   the Constructor
*******************************/

public Drbac()
{
  // Connecting to the database.
  try
  {
    Class.forName(driver);
    conn = DriverManager.getConnection(url, userid, pass);
    stmt = conn.createStatement();
  }
  catch(ClassNotFoundException cnfe)
  {
    System.err.println(cnfe);
  }
  catch(SQLException sqle)
  {
    System.err.println(sqle);  // error connection to database
  }
}
/*****************************************
   Public Methods:

    boolean      validateUser(String username, String password)
    Vector       getRoles(String user)
    String       getRole()
    Vector       getDatabases( String role)
    Vector       getTables( String role, String database)
    Vector       getTableFields( String role, String table)
    Vector       getTableFieldOperations( String role, String tablefield)
*****************************************/
/*****************************************
   this method checks the validity of
   the username and passord provided
*****************************************/
public boolean validateUser(String username, String password)
{
  try
  {
     stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                 ResultSet.CONCUR_READ_ONLY);
     rset = stmt.executeQuery("select * from users " +
         "where users.username = '" + username + "' " +
```

```
        "and users.password = '" + password + "'");

      if (rset.next())
      {
        validUser=true;
        return validUser;
      }
    }
    catch(SQLException sqle)
    {
      validUser=false;
    }
    validUser=false;
    return validUser;
  }

/*********************************************
   This method gets the roles that the user
   is allowed to assumein the application.
 *********************************************/
 public Vector getRoles(String user)
 {
    Vector roles = new Vector();
    try
    {
      stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                  ResultSet.CONCUR_READ_ONLY);
      rset = stmt.executeQuery("select role from user_role " +
         "where user_role.username = '" + user + "'");
      while (rset.next())
      {
        String tempRole = rset.getString(1);
        roles.addElement(tempRole);
      }
    }
    catch(SQLException sqle) {}
    return roles;
 }

 /************************************
   this method returns the active role
   selected by the user.
 ************************************/
 public String getRole()
 {
   return sRole;
```

```
}
/*******************************************
   This method allows to find the databases
   that are allowed for the assumed role.
*******************************************/
public Vector getDatabases( String role)
{
  Vector databases = new Vector();
  sRole=role;
  try
  {
     stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);
     rset = stmt.executeQuery("select database from role_database " +
         "where role_database.role = '" + sRole + "'");
     while (rset.next())
     {
        String tempDatabase = rset.getString(1);
        databases.addElement(tempDatabase);
     }
     rset.close();
  }
  catch(SQLException sqle){}
  return databases;
}

/*******************************************
   This method allows to find the tables of the
   database that the role is authorized to access
*******************************************/
 public Vector getTables( String role, String database)
 {
    Vector tables = new Vector();
    sRole=role;
    sDatabase=database;
    try
    {
       stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);
       rset = stmt.executeQuery("select tables from role_database_tables" +
         "where role_database_tables.database = '" + sDatabase + "'"+
         "AND role_database_tables.role = '" + sRole + "'" );
       while (rset.next())
       {
         String tempTable = rset.getString(1);
         tables.addElement(tempTable);
```

```
      }
    rset.close();
    }
    catch(SQLException sqle)  {}
    return tables;
}

/*******************************************
  This method allows to get the tablefields of the selected
 table in the database that the role is authorized to access
*******************************************/
public Vector getTableFields( String role, String table)
{
   Vector tableFields = new Vector();
   sRole=role;
   sTable=table;
   try
   {
      stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                              ResultSet.CONCUR_READ_ONLY);
      rset = stmt.executeQuery("select tablefield from table_tablefields"+
        "where table_tablefields.table = '" + sTable + "'");
      while (rset.next())
      {
        String tempTableField = rset.getString(1);
        tableFields.addElement(tempTableField);
      }
    rset.close();
    }
    catch(SQLException sqle) {}
    return tableFields;
}

/*******************************************
  This method allows to get the Operations
  that the active role is allowed to perform
  on the tableField of the selected table.
*******************************************/
public Vector getTableFieldOperations( String role, String tablefield)
{
   Vector tableFieldOperations = new Vector();
   sRole = role;
   sTableField=tablefield;
   try
   {
      stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
```

```
                                  ResultSet.CONCUR_READ_ONLY);
          rset = stmt.executeQuery("select operation from "+
           "role_tablefield_operation " +
           "where role_tablefield_operation.role = '" + sRole + "' "+
           "and role_tablefield_operation.tablefield = '" + sTableField+ "'");

          while (rset.next())
          {
            String ops =  new String(rset.getString(1));
            if (ops.equals("R"))   tableFieldOperations.addElement("Read");
            if (ops.equals("W")) tableFieldOperations.addElement("Write");
            if (ops.equals("RW"))
            {
              tableFieldOperations.addElement("Read");
              tableFieldOperations.addElement("Write");
            }
          }
          rset.close();
        }
        catch(SQLException sqle) {}
        return tableFieldOperations;
      }
    }
```

```
//-----------------------------------------------------------
// Filename:    RBACTreeStructure.java
// Date:        02/24/2002
// Project:     JDRBAC implementation
// Compiler:    SDK 1.3
//-----------------------------------------------------------
package rbactreetest;

import java.util.*;

/*
 *  This class allows the creation of the JDRBAC tree structure
 *  from the root to all possible leaves
 *
 * @authors Greg Nygard; Hammoudi Faouzi
 *
 */

public class RBACTreeStructure
{

  /******************************
    Data members
  ******************************/
  private TreeNode rootPtr;
  private TreeNode currentPtr;
  private TreeNode rolePtr;
  private TreeNode databasePtr;
  private TreeNode tablePtr;
  private TreeNode tablefieldPtr;
  private TreeNode operationPtr;

  public Vector roles;
  public Vector databases;
  public Vector tables;
  public Vector tablefields;
  public Vector operations;

  Drbac getinfo;
  Login userlogin;

  /******************************
     the Constructor
  ******************************/
  public RBACTreeStructure(Drbac getInfo, Login userLogin)
  {
```

```
  getinfo = getInfo;
  userlogin = userLogin;
  createTree();
}

/****************************************
   Public Methods:

    void        createTree()
    void        makeRoleChildren(TreeNode tempPtr)
    void        makeDatabaseChildren(TreeNode tempPtr, String roleName)
    void        makeTableChildren(TreeNode tempPtr, String roleName)
    void        makeTableFieldChildren(TreeNode tempPtr, String roleName)
    TreeNode      createNode(String name, TreeNode parent, TreeNode peer,
                   TreeNode child)
    Vector       getRoles()
    Vector       getDatabases(String roleName)
    Vector       getTables(String roleName, String dbName)
    Vector       getTableFields(String roleName, String dbName,
                 String tableName)
    Vector       getOperations(String roleName, String dbName,
                 String tableName, String tablefieldName)
    class       TreeNode
****************************************/

/************************************
   this method creates the the structure
   of the JDRBAC tree
************************************/
public void createTree()
{

  rootPtr = createNode("RBACTree", null, null, null);
  int marker = 0;
  roles = getinfo.getRoles(userlogin.getUserName());
  rootPtr.childPtr = createNode((String)roles.elementAt(marker),rootPtr,null,null);
  rolePtr = rootPtr.childPtr;
  currentPtr = rolePtr;
  while (marker < roles.size()-1)
  {
    marker = marker + 1;
    currentPtr.peerPtr = createNode((String)roles.elementAt(marker),null,null,null);
    currentPtr = currentPtr.peerPtr;
  }

  while (rolePtr != null)
```

```
    {
      makeRoleChildren(rolePtr);
      rolePtr = rolePtr.peerPtr;
    }
    rolePtr = rootPtr.childPtr;
  }

 /***********************************
     this method creates the children
     of node role
  ***********************************/
  public void makeRoleChildren(TreeNode tempPtr)
  {
    int marker = 0;
    String roleName = (String)tempPtr.nodeName;
    databases = getinfo.getDatabases(roleName);
    tempPtr.childPtr =
createNode((String)databases.elementAt(marker),tempPtr,null,null);
    databasePtr = tempPtr.childPtr;
    currentPtr = databasePtr;
    while (marker < databases.size()-1)
    {
      marker = marker + 1;
      currentPtr.peerPtr =
createNode((String)databases.elementAt(marker),null,null,null);
      currentPtr = currentPtr.peerPtr;
    }

    while (databasePtr != null)
    {
      makeDatabaseChildren(databasePtr, roleName);
      databasePtr = databasePtr.peerPtr;
    }
    databasePtr = tempPtr.childPtr;
  }

  /***********************************
     this method creates the children
     nodes of the database node.
  ***********************************/
  public void makeDatabaseChildren(TreeNode tempPtr, String roleName)
  {
    int marker = 0;
    tables = getinfo.getTables(roleName,tempPtr.nodeName);
    tempPtr.childPtr =
           createNode((String)tables.elementAt(marker),tempPtr,null,null);
```

```
  tablePtr = tempPtr.childPtr;
  currentPtr = tablePtr;
  while (marker < tables.size()-1) {
    marker = marker + 1;
    currentPtr.peerPtr =
          createNode((String)tables.elementAt(marker),null,null,null);
    currentPtr = currentPtr.peerPtr;
  }
  while (tablePtr != null)
  {
    makeTableChildren(tablePtr, roleName);
    tablePtr = tablePtr.peerPtr;
  }
  tablePtr = tempPtr.childPtr;
}

/************************************
   this method creates the children
   nodes ofthe table node
************************************/
public void makeTableChildren(TreeNode tempPtr, String roleName)
{
  int marker = 0;
  tablefields = getinfo.getTableFields(roleName, tempPtr.nodeName);
  tempPtr.childPtr =
      createNode((String)tablefields.elementAt(marker),tempPtr,null,null);
  tablefieldPtr = tempPtr.childPtr;
  currentPtr = tablefieldPtr;
  while (marker < tablefields.size()-1)
  {
    marker = marker + 1;
    currentPtr.peerPtr =
       createNode((String)tablefields.elementAt(marker),null,null,null);
    currentPtr = currentPtr.peerPtr;
  }
  while (tablefieldPtr != null)
  {
    makeTableFieldChildren(tablefieldPtr, roleName);
    tablefieldPtr = tablefieldPtr.peerPtr;
  }
  tablefieldPtr = tempPtr.childPtr;
}

/************************************
   this method creates the chilren
   nodes of the tablefield node
```

```
************************************/
public void makeTableFieldChildren(TreeNode tempPtr, String roleName)
{
  int marker = 0;
  operations = getinfo.getTableFieldOperations(roleName,tempPtr.nodeName);
  if (operations.size() == 0)
    tempPtr.childPtr = createNode("Read",tempPtr,null,null);
  else
    tempPtr.childPtr =
      createNode((String)operations.elementAt(marker),tempPtr,null,null);
  operationPtr = tempPtr.childPtr;
  currentPtr = operationPtr;
  while (marker < operations.size()-1)
  {
    marker = marker + 1;
    currentPtr.peerPtr =
        createNode((String)operations.elementAt(marker),null,null,null);
    currentPtr = currentPtr.peerPtr;
  }
}

/************************************
   this method creates the nodes
   of the JDRBAC tree
************************************/
public TreeNode createNode(String name, TreeNode parent, TreeNode peer,
                  TreeNode child)
{
   TreeNode newNode = new TreeNode();
   newNode.parentPtr = parent;
   newNode.peerPtr = peer;
   newNode.childPtr = child;
   newNode.nodeName = name;
   return newNode;
}

/****************************************
   this method fills the role node of the
   JDRBAC tree with the reference data
****************************************/
public Vector getRoles()
{
  Vector roleVector = new Vector();
  rolePtr = rootPtr.childPtr;
  while (rolePtr != null)
  {
```

```
    roleVector.addElement(rolePtr.nodeName);
    rolePtr = rolePtr.peerPtr;
  }
  return roleVector;
}

/****************************************
   this method fills the role node of the
   JDRBAC tree with the reference data
****************************************/
public Vector getDatabases(String roleName) {
  Vector databaseVector = new Vector();
  rolePtr = rootPtr.childPtr;
  while (rolePtr.nodeName != roleName) {
    rolePtr = rolePtr.peerPtr;
  }
  databasePtr = rolePtr.childPtr;
  while (databasePtr != null) {
    databaseVector.addElement(databasePtr.nodeName);
    databasePtr = databasePtr.peerPtr;
  }
  return databaseVector;
}

/****************************************
   this method fills the role node of the
   JDRBAC tree with the reference data
****************************************/
public Vector getTables(String roleName, String dbName)
{
  Vector tableVector = new Vector();
  rolePtr = rootPtr.childPtr;
  while (rolePtr.nodeName != roleName)
  {
    rolePtr = rolePtr.peerPtr;
  }
  databasePtr = rolePtr.childPtr;
  while (databasePtr.nodeName != dbName)
  {
    databasePtr = databasePtr.peerPtr;
  }
  tablePtr = databasePtr.childPtr;
  while (tablePtr != null)
  {
    tableVector.addElement(tablePtr.nodeName);
    tablePtr = tablePtr.peerPtr;
```

```
    }
    return tableVector;
  }

  /****************************************
     this method fills the role node of the
     JDRBAC tree with the reference data
  ****************************************/
 public Vector getTableFields(String roleName, String dbName, String tableName)
 {
    Vector tablefieldVector = new Vector();
    rolePtr = rootPtr.childPtr;
    while (rolePtr.nodeName != roleName)
    {
      rolePtr = rolePtr.peerPtr;
    }
    databasePtr = rolePtr.childPtr;
    while (databasePtr.nodeName != dbName)
    {
      databasePtr = databasePtr.peerPtr;
    }
    tablePtr = databasePtr.childPtr;
    while (tablePtr.nodeName != tableName)
    {
      tablePtr = tablePtr.peerPtr;
    }
    tablefieldPtr = tablePtr.childPtr;
    while (tablefieldPtr != null)
    {
      tablefieldVector.addElement(tablefieldPtr.nodeName);
      tablefieldPtr = tablefieldPtr.peerPtr;
    }
    return tablefieldVector;
  }

  /****************************************
     this method fills the role node of the
     JDRBAC tree with the reference data
  ****************************************/
  public Vector getOperations(String roleName, String dbName, String tableName,
    String tablefieldName)
  {
    Vector operationVector = new Vector();
    rolePtr = rootPtr.childPtr;
    while (rolePtr.nodeName != roleName)
    {
```

```
      rolePtr = rolePtr.peerPtr;
    }
    databasePtr = rolePtr.childPtr;
    while (databasePtr.nodeName != dbName)
    {
      databasePtr = databasePtr.peerPtr;
    }
    tablePtr = databasePtr.childPtr;
    while (tablePtr.nodeName != tableName)
    {
      tablePtr = tablePtr.peerPtr;
    }
    tablefieldPtr = tablePtr.childPtr;
    while (tablefieldPtr.nodeName != tablefieldName)
    {
      tablefieldPtr = tablefieldPtr.peerPtr;
    }
    operationPtr = tablefieldPtr.childPtr;
    while (operationPtr != null)
    {
      operationVector.addElement(operationPtr.nodeName);
      operationPtr = operationPtr.peerPtr;
    }
    return operationVector;
  }

 /****************************************
     this method fills the role node of the
     JDRBAC tree with the reference data
  ****************************************/
  public class TreeNode
  {
    public TreeNode parentPtr;
    public TreeNode peerPtr;
    public TreeNode childPtr;
    public String nodeName;
  }
}
```

```
//-------------------------------------------------------
// Filename:    RBACQueries.java
// Date:        02/24/2002
// Project:     JDRBAC implementation
// Compiler:    SDK 1.3
//-------------------------------------------------------
package rbactreetest;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import java.io.*;
import java.sql.*;

/*
 *  This class is used to execute and display the results of the queries
 *  that the user wants to perform on the selected database.
 *  This is only a prototype, and further enhancements or approach
 *  are possible and customizable.
 *
 * @authors Greg Nygard; Hammoudi Faouzi
 *
 */

public class RBACQueries extends JFrame
{
  /*******************************
    Data members
  *******************************/

  // ---------  Parameters for JDBC connection to Oracle------------
  String oracleuserid = "system"; //Login Name
  String pass = "manager"; //Password
  String oracleurl = "jdbc:oracle:thin:@rbac-server:1521:ATLANTIS";
  String oracledriver = "oracle.jdbc.driver.OracleDriver";
  // ---------  End of Parameters for JDBC connection to Oracle------

  // --------- Parameters for JDBC connection to DB2 Database -------
  String db2userid = "db2admin";
  String db2pass = "";
  // Password is the same as for Oracle
  String db2url = "jdbc:db2:Carthage";
  String db2driver = "COM.ibm.db2.jdbc.app.DB2Driver";
  // --------- End of Parameters for JDBC connection to DB2 Database -------
```

```
String temp;
Connection conn;
Statement state;
ResultSet rset;
JTextArea displayText = new JTextArea(100,50);

/******************************
   the Constructor
******************************/
public RBACQueries()
{
  super("RBAC Queries");

  Container content = getContentPane();

  JPanel displayPanel = new JPanel();
  displayPanel.add(displayText);
  content.add(displayPanel, BorderLayout.CENTER);

  JPanel buttonPanel = new JPanel();
  JButton close = new JButton("Close");
  close.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent event) {
     closeWindow();
   }
  });
  buttonPanel.add(close);
  content.add(buttonPanel, BorderLayout.SOUTH);
  setSize(600,400);
  setLocation(100,100);
  setVisible(true);
}
public void closeWindow()
{
 super.dispose();
}

/*****************************************
   Public Methods:
    void        doQueries(Object[] path)
*****************************************/

/***********************************
   this method executes the query submitted
   by the user of the application.
***********************************/
```

```
public void doQueries(Object[] path)
{
  String role = path[1].toString();
  String database = path[2].toString();
  String table = path[3].toString();
  String tablefield = path[4].toString();
  String driver;
  String url;
  String userid;
  if (database.equalsIgnoreCase("Atlantis") )
  {
    System.out.println("Connecting to Atlantis");
    driver = oracledriver;
    url = oracleurl;
    userid = oracleuserid;
    try
    {
      Class.forName(driver);
        conn = DriverManager.getConnection(url, userid, pass);
      state = conn.createStatement();
    }
    catch(ClassNotFoundException cnfe)
    {
      System.out.println("No connection");
    }
    catch(SQLException sqle)
    {
      System.out.println("No connection");
    }
  }
  else
  {
    System.out.println("Connecting to Carthage");
    driver = db2driver;
    url = db2url;
    userid = ""; //db2userid;
    try
    {
      Class.forName(driver);
        conn = DriverManager.getConnection(url);
      state = conn.createStatement();
    }
    catch(ClassNotFoundException cnfe)
    {
      System.out.println("No connection");
    }
```

```
      catch(SQLException sqle)
      {
         System.out.println("No connection");
      }
    }
    displayText.append("Database: "+database+"\n");
    displayText.append("Table: "+table+"\n");
    displayText.append("TableField: "+tablefield+"\n");
    displayText.append("Results: \n");
    try
    {
       state = conn.createStatement();
       rset = state.executeQuery("select * from "+table);
       while (rset.next())
       {
         displayText.append(rset.getString(1)+" ");
         System.out.println(rset.getString(1));
       }
       displayText.append("\n");
       rset.close();
       state.close();
    }
    catch(SQLException sqle)
    {
      System.out.println("No return from database");
    }
  }
}
```

```
//------------------------------------------------------
// Filename:    RBACTree.java
// Date:        02/24/2002
// Project:     JDRBAC implementation
// Compiler:    SDK 1.3
//------------------------------------------------------
package rbactreetest;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

/*
 *  This class is used to display the JDRBAC
 *  tree interface and the selection made by
 *  the user.
 *
 * @authors Greg Nygard; Hammoudi Faouzi
 *
 */

public class RBACTree extends JFrame
{

 /*******************************
   Data members
  *******************************/

 static RBACTree rbactree;
 RBACQueries rbacqueries;
 static Drbac getInfo = new Drbac();
 static RBACTreeStructure treeStructure;
 static Login userLogin = new Login();
 static boolean loginTest;

 /*******************************
    Main of the application
 *******************************/

 public static void main(String[] args)
 {
  loggingIn();  // Invokes the login interface
  // tree construction
```

```
  treeStructure = new RBACTreeStructure(getInfo, userLogin);
  rbactree = new RBACTree();  // Tree display
}

private Icon customOpenIcon = new ImageIcon("c:/icons/openbook.gif");
private Icon customClosedIcon = new ImageIcon("c:/icons/closedbook.gif");
private Icon customLeafIcon = new ImageIcon("c:/icons/viewdoc.gif");

String selection = new String();
Object[] path;
JTextArea selectText = new JTextArea(18,25);
TreePath tp;
JRadioButton read = new JRadioButton("Read",false);
JRadioButton write = new JRadioButton("Write",false);

/*******************************
    the Constructor
*******************************/
public RBACTree() {
  super("RBAC Tree");
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) { System.exit(0); }
  });
  Container content = getContentPane();

  String userName = userLogin.getUserName();
  DefaultMutableTreeNode root = createNodes(userName);
  JTree tree = new JTree(root);
  DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
  renderer.setOpenIcon(customOpenIcon);
  renderer.setClosedIcon(customClosedIcon);
  renderer.setLeafIcon(customLeafIcon);
  tree.setCellRenderer(renderer);
  tree.collapseRow(1);
  tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent tse) {
      tp = tse.getNewLeadSelectionPath();
      if (tp.getPathCount() == 5) {
        setOperations(tp.getPath());
      }
      else {
        read.setSelected(true);
        read.setEnabled(false);
        write.setSelected(false);
        write.setEnabled(false);
      }
```

```
    }
  });
  content.add(new JScrollPane(tree), BorderLayout.CENTER);
  JPanel opPanel = new JPanel();
  opPanel.setLayout(new GridLayout(6,1));
  read.setEnabled(false);
  read.setSelected(false);
  write.setEnabled(false);
  write.setSelected(false);
  final ButtonGroup bgroup = new ButtonGroup();
  bgroup.add(read);
  bgroup.add(write);
  opPanel.add(new JLabel());
  opPanel.add(new JLabel("Operations"));
  opPanel.add(read);
  opPanel.add(write);
  content.add(opPanel, BorderLayout.WEST);
  content.add(new JLabel("                              Data Options    " +
  "                         Data Selections"), BorderLayout.NORTH);
  JPanel buttonPanel = new JPanel();
  buttonPanel.add(new JLabel("                 "));
  JButton select = new JButton("Select");
  select.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      if (tp.getPathCount() == 5) {
        selectText.setText("");
        path = tp.getPath();
        String role = path[1].toString();
        String database = path[2].toString();
        String table = path[3].toString();
        String tablefield = path[4].toString();
        selection = role+" - "+database+" - "+table+" - "+tablefield;
        if (read.isSelected()) {
          selection = selection + " - read";
          selectText.append(selection + "\n");
        }
        if (write.isSelected()) {
          selection = selection + " - write";
          selectText.append(selection + "\n");
        }
      }
    }
  });
  buttonPanel.add(select);
  JButton clear = new JButton("Clear");
  clear.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent event) {
      selectText.setText("");
    }
  });
  buttonPanel.add(clear);
  buttonPanel.add(new JLabel("                              "));
  JButton submit = new JButton("Submit");
  submit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      rbacqueries = new RBACQueries();
      rbacqueries.doQueries(path);
    }
  });
  buttonPanel.add(submit);
  content.add(buttonPanel, BorderLayout.SOUTH);
  JPanel entryPanel = new JPanel();
  entryPanel.setLayout(new FlowLayout());
  entryPanel.add(new JScrollPane(selectText));
  content.add(entryPanel, BorderLayout.EAST);
  setSize(600,400);
  setLocation(100,100);
  setVisible(true);
}

private static void loggingIn()
{
  loginTest = userLogin.validLogon();
  while (!loginTest)
  {
    loginTest = userLogin.validLogon();
  }
  userLogin.dispose();
}

private DefaultMutableTreeNode createNodes(String userName)
{
  DefaultMutableTreeNode root =
    new DefaultMutableTreeNode(userName);
  DefaultMutableTreeNode role;
  DefaultMutableTreeNode database;
  DefaultMutableTreeNode table;
  DefaultMutableTreeNode tablefield;
  DefaultMutableTreeNode operation;

  Vector userRole = new Vector();
  userRole = treeStructure.getRoles();
```

```
  for (int x = 0; x < userRole.size(); x++)
  {
    role = new DefaultMutableTreeNode((String)userRole.elementAt(x));
    root.add(role);
    createDatabaseNodes(role);
  }
  return(root);
}

private void createDatabaseNodes(DefaultMutableTreeNode role)
{
  DefaultMutableTreeNode database;
  Vector databases = new Vector();
  String roleName = role.toString();
  databases = treeStructure.getDatabases(roleName);
  for (int x = 0; x < databases.size(); x++)
  {
    database = new DefaultMutableTreeNode((String)databases.elementAt(x));
    role.add(database);
    createTableNodes(database, roleName);
  }
}

private void createTableNodes(DefaultMutableTreeNode database,
          String roleName)
{

  DefaultMutableTreeNode table;
  Vector tables = new Vector();
  String dbName = database.toString();
  tables = treeStructure.getTables(roleName,dbName);

  for (int x = 0; x < tables.size(); x++)
  {
    table = new DefaultMutableTreeNode((String)tables.elementAt(x));
    database.add(table);
    createTableFieldNodes(table,roleName,dbName);
  }
}

private void createTableFieldNodes(DefaultMutableTreeNode table,
         String roleName, String dbName)
{
  DefaultMutableTreeNode tablefield;
  Vector tablefields = new Vector();
```

```
    String tableName = table.toString();
    tablefields = treeStructure.getTableFields(roleName,dbName,tableName);

    for (int x = 0; x < tablefields.size(); x++)
    {
      tablefield =
          new DefaultMutableTreeNode((String)tablefields.elementAt(x));
      table.add(tablefield);
    }
  }

  private void setOperations(Object[] path)
  {
    Vector operations = new Vector();
    String role = path[1].toString();
    String database = path[2].toString();
    String table = path[3].toString();
    String tablefield = path[4].toString();
    operations = treeStructure.getOperations(role,database,table,tablefield);
    for (int x = 0; x < operations.size(); x++)
    {
      if ((String)operations.elementAt(x) == "Read")
      {
        read.setEnabled(true);
        if (x < 1)
        {
          write.setEnabled(false);
          read.setSelected(true);
        }
      }
      if ((String)operations.elementAt(x) == "Write")
      {
        write.setEnabled(true);
        if (x < 1)
        {
          read.setEnabled(false);
          write.setSelected(true);
        }
      }
    }
  }
}
```

```
//------------------------------------------------------
// Filename:    Jdrbac.java
// Date:        02/20/2002
// Compiler:    SDK 1.3
//------------------------------------------------------

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/*
 *  This class represents the administration Interface available
 *  for the JDRBAC administrator to perform his duties.
 *  This interface allows to alleviate the requiremment for the
 *  administrator to be expert in SQL. It is a menu driven application
 *  that can be used as tools instead of writing SQL queries
 *
 * @authors Greg Nygard; Faouzi Hammoudi
 *
 */

public class Jdrbac extends JFrame implements ActionListener
{

  /******************************
          Data members
  ******************************/

  JMenuBar mb;          // a MenuBar to hold the menus

  private JMenu users, roles,databases,tables,tablefields,operations,exit;

  // menu items for the user menu
  private JMenuItem addUser, removeUser, changeUserPwd, showUsers;

  // Menu Items for the roles
  private JMenuItem addRole,removeRole,showRoles;

  // Menu Items for the databases
  private JMenuItem addDatabase,removeDatabase,showDatabases;

  // Menu Items for the tables
  private JMenuItem addTable,removeTable,showTables;

  // Menu Items for the tablefields
  private JMenuItem addTablefield,removeTablefield,showTablefields;
```

```
// Menu Items for the operations
private JMenuItem addOperation,removeOperation,showOperations;

// Menu Items for the Exit menu
private JMenuItem exitApp;


/******************************
        Constructor
******************************/

public Jdrbac()
{
  super(" JDRBAC: Java Database Role-Based Access Control Web Application");
  addWindowListener(new WindowAdapter( ) {       // to catch the
  public void windowClosing(WindowEvent e) {   // closing window
    System.exit(0);
    }
  });

  setSize(700, 450);      // sets the size of the window

  setLocation(50, 50);   // sets the location of the window
  addMenus();
  setVisible(true);
}


/******************************
Public Methods:

  void   actionPerformed(ActionEvent evt)
  void   addMenus()
  void   main()

******************************/

public void addMenus()     // method to add menus to a menu bar
{

   mb = new JMenuBar();  // create a menu bar

   setJMenuBar(mb);      // adds the menubar to the window

   // The Users Management menu
```

```
users = new JMenu("Users Mgmt");  // create the menu for the line width
users.setFont(new Font("TimesRoman",Font.BOLD, 14));

addUser = new JMenuItem("Add User");
addUser.setFont(new Font("TimesRoman",Font.BOLD, 12));

removeUser = new JMenuItem("Remove User");
removeUser.setFont(new Font("TimesRoman",Font.BOLD, 12));

changeUserPwd = new JMenuItem("Change User Password");
changeUserPwd.setFont(new Font("TimesRoman",Font.BOLD, 12));

showUsers = new JMenuItem("Show users");
showUsers.setFont(new Font("TimesRoman",Font.BOLD, 12));

users.add(addUser);        //  add the menu item to the menu
users.addSeparator();        //  add a line separator
addUser.addActionListener(this); //  add the listener to the menu item

users.add(removeUser);       // same thing for the other menu items
users.addSeparator();
removeUser.addActionListener(this);
users.add(changeUserPwd);
users.addSeparator();
changeUserPwd.addActionListener(this);
users.add(showUsers);
showUsers.addActionListener(this);
mb.add(users);

// The roles menu
roles = new JMenu("Roles Mgmt");
roles.setFont(new Font("TimesRoman",Font.BOLD, 14));
addRole = new JMenuItem("Add Role");
addRole.setFont(new Font("TimesRoman",Font.BOLD, 12));
removeRole = new JMenuItem("Remove Role");
removeRole.setFont(new Font("TimesRoman",Font.BOLD, 12));
showRoles = new JMenuItem("Show Roles");
showRoles.setFont(new Font("TimesRoman",Font.BOLD, 12));
roles.add(addRole);
roles.addSeparator();
addRole.addActionListener(this);
roles.add(removeRole);
roles.addSeparator();
removeRole.addActionListener(this);
roles.add(showRoles);
showRoles.addActionListener(this);
```

```java
// add the menu to the menu bar
mb.add(roles);

 // The database menu
databases = new JMenu("Databases");
databases.setFont(new Font("TimesRoman",Font.BOLD, 14));

addDatabase = new JMenuItem("Add Database-Role");
addDatabase.setFont(new Font("TimesRoman",Font.BOLD, 12));

removeDatabase = new JMenuItem("Remove Database-Role");
removeDatabase.setFont(new Font("TimesRoman",Font.BOLD, 12));

showDatabases = new JMenuItem("Show Databases");
showDatabases.setFont(new Font("TimesRoman",Font.BOLD, 12));

databases.add(addDatabase);
databases.addSeparator();
addDatabase.addActionListener(this);

databases.add(removeDatabase);
databases.addSeparator();
removeDatabase.addActionListener(this);
databases.add(showDatabases);
showDatabases.addActionListener(this);
mb.add(databases);
// The table menu
tables = new JMenu("Tables");
tables.setFont(new Font("TimesRoman",Font.BOLD, 14));
addTable = new JMenuItem("Add Database-Table");
addTable.setFont(new Font("TimesRoman",Font.BOLD, 12));
removeTable = new JMenuItem("Remove Database-Table");
removeTable.setFont(new Font("TimesRoman",Font.BOLD, 12));
showTables = new JMenuItem("Show Database-Tables");
showTables.setFont(new Font("TimesRoman",Font.BOLD, 12));
tables.add(addTable);
tables.addSeparator();
addTable.addActionListener(this);
tables.add(removeTable);
tables.addSeparator();
removeTable.addActionListener(this);
tables.add(showTables);
showTables.addActionListener(this);
mb.add(tables);
```

```
 // The tablefields menu
tablefields = new JMenu("Tablefields");
tablefields.setFont(new Font("TimesRoman",Font.BOLD, 14));
addTablefield = new JMenuItem("Add Table-Tablefield");
addTablefield.setFont(new Font("TimesRoman",Font.BOLD, 12));
removeTablefield = new JMenuItem("Remove Table-Tablefield");
removeTablefield.setFont(new Font("TimesRoman",Font.BOLD, 12));
showTablefields = new JMenuItem("Show Table-Tablefields");
showTablefields.setFont(new Font("TimesRoman",Font.BOLD, 12));
tablefields.add(addTablefield);
tablefields.addSeparator();
addTablefield.addActionListener(this);
tablefields.add(removeTablefield);
tablefields.addSeparator();
removeTablefield.addActionListener(this);
tablefields.add(showTablefields);
showTablefields.addActionListener(this);
mb.add(tablefields);
 // The Operations menu
operations = new JMenu("Operations");
operations.setFont(new Font("TimesRoman",Font.BOLD, 14));
addOperation = new JMenuItem("Add Tablefield-Operation");
addOperation.setFont(new Font("TimesRoman",Font.BOLD, 12));
removeOperation = new JMenuItem("Remove Tablefield-Operation");
removeOperation.setFont(new Font("TimesRoman",Font.BOLD, 12));
showOperations = new JMenuItem("Show Tablefield-Operations");
showOperations.setFont(new Font("TimesRoman",Font.BOLD, 12));
operations.add(addOperation);
operations.addSeparator();
addOperation.addActionListener(this);
operations.add(removeOperation);
operations.addSeparator();
removeOperation.addActionListener(this);
operations.add(showOperations);
showOperations.addActionListener(this);
mb.add(operations);
// The Exit menu
exit = new JMenu("Exit");
exit.setFont(new Font("TimesRoman",Font.BOLD, 14));
exitApp = new JMenuItem("Exit the application");
exitApp.setFont(new Font("TimesRoman",Font.BOLD, 12));
exit.add(exitApp);
exitApp.addActionListener(this);
mb.add(exit);

}
```

```
  /***************************************************

  Method actionPerformed: defines the actions to take
  for the button various selections in the two menus

  ****************************************************/

public void actionPerformed(ActionEvent evt)
  {
     if(evt.getSource()==exitApp)
     {
       System.exit(0);      // exit the application;
     }
     if(evt.getSource()==addUser)
     {
       new AddUser(this);
     }
     if(evt.getSource()==removeUser)
     {
       new RemoveUser(this);
     }
     if(evt.getSource()==showUsers)
     {
       new ShowUsers(this);
     }
     if(evt.getSource()==addRole)
     {
       new AddRole(this);
     }
     if(evt.getSource()==removeRole)
     {
       new RemoveRole(this);
     }
     if(evt.getSource()==showRoles)
     {
       new ShowRoles(this);
     }
     if(evt.getSource()==addUserRole)
     {
       new AddUserRole(this);
     }
     if(evt.getSource()==removeUserRole)
     {
       new RemoveUserRole(this);
     }
```

```
        if(evt.getSource()==showUserRoles)
    {
      new ShowUserRoles(this);
    }
     if(evt.getSource()==showUsersRoles)
    {
      new ShowUsersRoles(this);
    }
     if(evt.getSource()==addRoleDatabase)
    {
      new AddRoleDatabase(this);
    }
     if(evt.getSource()==removeRoleDatabase)
    {
      new RemoveRoleDatabase(this);
    }
     if(evt.getSource()==showRoleDatabases)
    {
      new ShowRoleDatabase(this);
    }
      }
  public static void main(String args[])
  {
    Jdrbac nj = new Jdrbac();

  }
}// End of Jdrbac.java
```

```
//------------------------------------------------------
// Filename:    AddUser.java
// Date:        02/20/2002
// Compiler:    SDK 1.3
//------------------------------------------------------

import java.sql.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.Vector;

/*
 *  This class allows to add a user to the JDRBAC application
 *  It allows to insert into the users table a record that
 *  represents the added user
 *
 * @authors Greg Nygard; Faouzi Hammoudi
 *
 */

public class AddUser extends JFrame implements ActionListener
{

  /*******************************
         Data members
  *******************************/
  private JTextField nameField;
  private JPasswordField pwdField;
  private JTextField status;
  public static boolean validUser=true;
  private String temp;
  private String username;
  private String password;
  private JButton addUser, clear, exit;
  private JPanel pN, pNs;
  Container container;      // a generic container
  private ResultSet rset;
  private Statement stmt;
  Jdrbac parent;

  // ----------- -----  Connection Variables ------------------

  String userid = "sa";
  String pass = "";
  String url = "jdbc:hsqldb:hsql://localhost";
  String driver = "org.hsqldb.jdbcDriver";
```

```
Connection conn;

/******************************
        Constructor
******************************/

public AddUser(Jdrbac parent)
{
  super("Jdrbac: Adding a User");
  this.parent=parent;
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
       System.exit(0);
    }
  });
  container = getContentPane();   // gets the contentPane for the frame

  container.setLayout(new BorderLayout()); // sets the layout manager to
                                   // BorderLayout
  pN = new JPanel();      pNs = new JPanel();
  nameField = new JTextField(20);
  pwdField = new JPasswordField(20);
  nameField.addActionListener(this);
  pwdField.addActionListener(this);
  pwdField.setEchoChar('*');
  pN.add(new Label("Enter the Username:"));
  pN.add(nameField);
  pN.add(new Label("and P A S S W O R D:"));
  pN.add(pwdField);
  addUser = new JButton("Add");
  addUser.addActionListener(this);
                clear = new JButton("Clear");
  clear.addActionListener(this);
  exit = new JButton("Exit");
  exit.addActionListener(this);
  pNs.add(addUser); pNs.add(clear);pNs.add(exit);
  container.add(pN,BorderLayout.CENTER);
  pN.add(new JLabel("Operation   S t a t u s:"));
  pN.add(status = new JTextField(20));
  container.add(pNs,BorderLayout.SOUTH);
  setBounds(250,150,380, 200);      // sets the size of the window
  setVisible(true);
```

```
  // Connecting to the reference data
  try
  {
    Class.forName(driver);
    conn = DriverManager.getConnection(url, userid, pass);
    stmt = conn.createStatement();
  }
  catch(ClassNotFoundException cnfe)
  {
    System.err.println(cnfe);
  }
  catch(SQLException sqle)
  {
    System.err.println(sqle);  // error connection to database
  }

}

public void actionPerformed(ActionEvent evt)
{
  Object source = evt.getSource();
                if(evt.getSource()==exit)
   {
      this.dispose();
   }
   if(evt.getSource()==clear)
   {
     nameField.setText("");
     pwdField.setText("");
                         status.setText("");
   }
  if(source.equals(addUser))
  {
    username = new String(nameField.getText());
    password = new String(pwdField.getText());
                  validateUser(username, password);
                  if(!validUser)
                  {
      try
      {
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_READ_ONLY);
        rset = stmt.executeQuery("insert into users " +
       "values ('" + username + "' " + "," + "'" + password + "'" + ")");

        status.setText("User added ");
```

```
            nameField.setText("");
            pwdField.setText("");
          }
          catch(SQLException sqle)
          {
          status.setText(sqle.getMessage());
          }
        }
        else status.setText("The user is already in the database");
      }

    }

    /****************************************************
    This method checks if the user to be added is not
    currently in the database. If yes, the administrator has
    assign a new username for the user to resolve the conflict.
    *****************************************************/

    public boolean validateUser(String username, String password)
    {
      try
      {
          stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                      ResultSet.CONCUR_READ_ONLY);
          rset = stmt.executeQuery("select * from users " +
            "where users.username = '" + username + "' " +
            "and users.password = '" + password + "'");

         if (rset.next())
         {
             validUser=true;
             return validUser;
          }
        }
        catch(SQLException sqle)
        {
          validUser=false;
        }
        validUser=false;
        return validUser;
     }
  }// End of AddUser.java
```

```
//-------------------------------------------------------
// Filename:    RemoveUser.java
// Date:        02/20/2002
// Compiler:    SDK 1.3
//-------------------------------------------------------

import java.sql.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

/*
 *  This class allows to remove a user
 *  from the JDRBAC application
 *
 * @authors Greg Nygard; Faouzi Hammoudi
 *
 */

public class RemoveUser extends JFrame implements ActionListener
{
  /********************************
         Data members
  ********************************/
  private JTextField nameField;
  private JPasswordField pwdField;
  private JTextField status;
  public static boolean validUser=true;
  private String temp;
  private String username;
  private String password;
  private JButton removeUser, clear, exit;
  private JPanel pN, pNs;
  Container container;      // a generic container
  private ResultSet rset;
  private Statement stmt;
  Jdrbac parent;

  // ----------- -----  Connection Variables ------------------
  String userid = "sa";
  String pass = "";
  String url = "jdbc:hsqldb:hsql://localhost";
  String driver = "org.hsqldb.jdbcDriver";
```

```
Connection conn;

/******************************
        Constructor
******************************/

public RemoveUser(Jdrbac parent)
{
  super("Jdrbac: Removing a user");
  this.parent=parent;
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
       System.exit(0);
    }
  });
  container = getContentPane();   // gets the contentPane for the frame

  container.setLayout(new BorderLayout()); // sets the layout manager to
                                  // BorderLayout
  pN = new JPanel();      pNs = new JPanel();
  nameField = new JTextField(20);
  pwdField = new JPasswordField(20);
  nameField.addActionListener(this);
  pwdField.addActionListener(this);
  pwdField.setEchoChar('*');
  pN.add(new Label("Enter the Username:"));
  pN.add(nameField);
  pN.add(new Label("and P A S S W O R D:"));
  pN.add(pwdField);
  removeUser = new JButton("Remove");
  removeUser.addActionListener(this);
              clear = new JButton("Clear");
  clear.addActionListener(this);
  exit = new JButton("Exit");
  exit.addActionListener(this);
  pNs.add(removeUser); pNs.add(clear);pNs.add(exit);
  container.add(pN,BorderLayout.CENTER);
  pN.add(new JLabel("Operation   S t a t u s:"));
  pN.add(status = new JTextField(20));
  container.add(pNs,BorderLayout.SOUTH);
  setBounds(250,150,380, 200);      // sets the size of the window
  setVisible(true);
  try
  {
```

```
      Class.forName(driver);
      conn = DriverManager.getConnection(url, userid, pass);
      stmt = conn.createStatement();
    }
    catch(ClassNotFoundException cnfe)
    {
      System.err.println(cnfe);
    }
    catch(SQLException sqle)
    {
      System.err.println(sqle);  // error connection to database
    }

  }

  public void actionPerformed(ActionEvent evt)
  {
    Object source = evt.getSource();
                    if(evt.getSource()==exit)
     {
        this.dispose();
     }
     if(evt.getSource()==clear)
     {
       nameField.setText("");
       pwdField.setText("");
                                status.setText("");
     }
    if(source.equals(removeUser))
    {
      username = new String(nameField.getText());
      password = new String(pwdField.getText());
                      validateUser(username, password);
                      if(validUser)
                      {
        try
        {
          stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                  ResultSet.CONCUR_READ_ONLY);
          rset = stmt.executeQuery("delete from users " +
            "where users.username = '" + username + "' " +
            "and users.password = '" + password + "'");

          status.setText("User removed ");
          nameField.setText("");
          pwdField.setText("");
```

```
        }
        catch(SQLException sqle)
        {
        status.setText(sqle.getMessage());
        }
      }
      else status.setText("The user is not in the database");
    }

  }

  /**************************************************
   This method checks if the user to be removed is not
   currently in the database.
   ***************************************************/
   public boolean validateUser(String username, String password)
   {
     try
     {
         stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);
         rset = stmt.executeQuery("select * from users " +
           "where users.username = '" + username + "' " +
           "and users.password = '" + password + "'");

        if (rset.next())
        {
            validUser=true;
            return validUser;
         }
      }
      catch(SQLException sqle)
      {
         validUser=false;
      }
      validUser=false;
      return validUser;
   }
}// End of RemoveUser.java
```

```
//-----------------------------------------------------
// Filename:    ShowUsers.java
// Date:        02/20/2002
// Compiler:    SDK 1.3
//-----------------------------------------------------
import java.sql.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

/*
 *  This class allows to see the list of the users of the JDRBAC
 *  application. It provides an easy way to query the database
 *  and find all the users.
 *
 * @authors Greg Nygard; Faouzi Hammoudi
 *
 */

public class ShowUsers extends JFrame implements ActionListener
{
 /*******************************
         Data members
  *******************************/
  private List usernames;
  private JTextField status;
  private JButton exit;
  private Panel pN, pNs;
  Container container;      // a generic container
  private ResultSet rset;
  private Statement stmt;
  Jdrbac parent;

  // ----------- -----  Connection Variables ------------------
  String userid = "sa";
  String pass = "";
  String url = "jdbc:hsqldb:hsql://localhost";
  String driver = "org.hsqldb.jdbcDriver";
  Connection conn;

  /*******************************
         Constructor
  *******************************/
```

```
public ShowUsers(Jdrbac parent)
{
  super("Jdrbac: List of the users and passwords");
  this.parent=parent;
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
       System.exit(0);
    }
  });
  container = getContentPane();   // gets the contentPane for the frame

  container.setLayout(new BorderLayout()); // sets the layout manager to
                                    // BorderLayout
  pN = new Panel(); pN.setLayout(new GridLayout(2,1));
              pNs = new Panel();
  usernames = new List(4, false);
  usernames.addActionListener(this);
  pN.add(usernames);
  pN.add(new Label("Username *** | *** Password:"));
  pN.add(usernames);
  exit = new JButton("Exit");
  exit.addActionListener(this);
  pNs.add(exit);
  container.add(pN,BorderLayout.CENTER);
  pNs.add(new JLabel("Operation   S t a t u s:"));
  pNs.add(status = new JTextField(20));
  container.add(pNs,BorderLayout.SOUTH);
  setBounds(250,150,430, 250);       // sets the size of the window
  setVisible(true);
  try
  {
    Class.forName(driver);
    conn = DriverManager.getConnection(url, userid, pass);
    stmt = conn.createStatement();
                  getUsers();
  }
  catch(ClassNotFoundException cnfe)
  {
    System.err.println(cnfe);
  }
  catch(SQLException sqle)
  {
    System.err.println(sqle);  // error connection to database
  }
```

```
  }

  public void actionPerformed(ActionEvent evt)
  {
    Object source = evt.getSource();
                    if(evt.getSource()==exit)
     {
        this.dispose();
     }
  }
  public void getUsers()
  {
    try
    {
      stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                  ResultSet.CONCUR_READ_ONLY);
      rset = stmt.executeQuery("select * from users ");

      while (rset.next())
      {
                            String temp= new String(rset.getString(1)+
                     " *** | *** "+  rset.getString(2));
        usernames.add(temp);
        status.setText("Users In the database ");
      }
      rset.close();
    }
    catch(SQLException sqle)
    {
    status.setText(sqle.getMessage());
    }
  }
}// End of ShowUsers.java
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. Dan Boger, Code C4I
   Naval Postgraduate School
   Monterey, CA 93943-5118

4. LCDR Chris Eagle, Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5118

5. Professor James Bret Michael, Code CS/Mj
   Naval Postgraduate School
   Monterey, CA 93943-5118

6. Professor John Osmundson, Code C4I
   Naval Postgraduate School
   Monterey, CA 93943-5118

7. Lieutenant Greg Nygard, Code 32
   Naval Postgraduate School
   Monterey, CA 93943-5118

8. Captain Faouzi Hammoudi, Code 32
   Naval Postgraduate School
   Monterey, CA 93943-5118

9. Dr. Ravi Sandhu
   George Mason University
   ISE Department, Mail Stop 4A4
   Fairfax, VA 22030-4444

10. Dr. Terry Mayfield
    IDA/CSED
    1801 N. Beauregard St.
    Alexandria, VA 22311

11. D. Richard Kuhn
National Institute of Standards and Technology
Computer Security Division
100 Bureau Drive Stop 8930
Gaithersburg, MD 20899-8930

12. Etat Major de l’Armée de l’Air
Ministere de la Defense Nationale
Avenue Bab M’nara,
Tunis 1030, Tunisia